In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
pio.templates.default = "plotly_white"
```

In [2]:
```
df = pd.read_csv('dynamic_pricing.csv')
```

In [3]:
```
df
```

Out[3]:

	Number_of_Riders	Number_of_Drivers	Location_Category	Customer_Loyalty_Status	Number_of_Past_Rides	Average_Ratings	Time_of_E
0	90	45	Urban	Silver	13	4.47	
1	58	39	Suburban	Silver	72	4.06	
2	42	31	Rural	Silver	0	3.99	A
3	89	28	Rural	Regular	67	4.31	A
4	78	22	Rural	Regular	74	3.77	A
...	...	...	...	...	...	...	
995	33	23	Urban	Gold	24	4.21	
996	84	29	Urban	Regular	92	4.55	
997	44	6	Suburban	Gold	80	4.13	
998	53	27	Suburban	Regular	78	3.63	
999	78	63	Rural	Gold	14	4.21	A

1000 rows × 10 columns

In [4]:
```
df.head()
```

Out[4]:

	Number_of_Riders	Number_of_Drivers	Location_Category	Customer_Loyalty_Status	Number_of_Past_Rides	Average_Ratings	Time_of_Boo
0	90	45	Urban	Silver	13	4.47	
1	58	39	Suburban	Silver	72	4.06	Ev
2	42	31	Rural	Silver	0	3.99	After
3	89	28	Rural	Regular	67	4.31	After
4	78	22	Rural	Regular	74	3.77	After

In [5]:
```
df.tail()
```

Out[5]:

	Number_of_Riders	Number_of_Drivers	Location_Category	Customer_Loyalty_Status	Number_of_Past_Rides	Average_Ratings	Time_of_E
995	33	23	Urban	Gold	24	4.21	
996	84	29	Urban	Regular	92	4.55	
997	44	6	Suburban	Gold	80	4.13	
998	53	27	Suburban	Regular	78	3.63	
999	78	63	Rural	Gold	14	4.21	A

In [6]:
```
df.describe()
```

Out[6]:

	Number_of_Riders	Number_of_Drivers	Number_of_Past_Rides	Average_Ratings	Expected_Ride_Duration	Historical_Cost_of_Ride
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000	1000.000000
mean	60.372000	27.076000	50.031000	4.257220	99.58800	372.502623
std	23.701506	19.068346	29.313774	0.435781	49.16545	187.158756
min	20.000000	5.000000	0.000000	3.500000	10.00000	25.993449
25%	40.000000	11.000000	25.000000	3.870000	59.75000	221.365202
50%	60.000000	22.000000	51.000000	4.270000	102.00000	362.019426
75%	81.000000	38.000000	75.000000	4.632500	143.00000	510.497504
max	100.000000	89.000000	100.000000	5.000000	180.00000	836.116419

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Number_of_Riders        1000 non-null   int64  
 1   Number_of_Drivers       1000 non-null   int64  
 2   Location_Category       1000 non-null   object 
 3   Customer_Loyalty_Status 1000 non-null   object 
 4   Number_of_Past_Rides    1000 non-null   int64  
 5   Average_Ratings         1000 non-null   float64
 6   Time_of_Booking         1000 non-null   object 
 7   Vehicle_Type            1000 non-null   object 
 8   Expected_Ride_Duration  1000 non-null   int64  
 9   Historical_Cost_of_Ride 1000 non-null   float64
dtypes: float64(2), int64(4), object(4)
memory usage: 78.3+ KB
```

In [8]:
```
df.columns
```

Out[8]:
```
Index(['Number_of_Riders', 'Number_of_Drivers', 'Location_Category',
       'Customer_Loyalty_Status', 'Number_of_Past_Rides', 'Average_Ratings',
       'Time_of_Booking', 'Vehicle_Type', 'Expected_Ride_Duration',
       'Historical_Cost_of_Ride'],
      dtype='object')
```

In [9]:
```
df.dtypes
```

Out[9]:
```
Number_of_Riders            int64
Number_of_Drivers           int64
Location_Category          object
Customer_Loyalty_Status    object
Number_of_Past_Rides        int64
Average_Ratings           float64
Time_of_Booking            object
Vehicle_Type               object
Expected_Ride_Duration      int64
Historical_Cost_of_Ride    float64
dtype: object
```

In [10]:
```
df.duplicated().sum()
```

Out[10]:
```
0
```

In [11]:
```
df.nunique()
```

Out[11]:
```
Number_of_Riders             81
Number_of_Drivers            79
Location_Category             3
Customer_Loyalty_Status       3
Number_of_Past_Rides        101
Average_Ratings             151
Time_of_Booking               4
Vehicle_Type                  2
Expected_Ride_Duration      171
Historical_Cost_of_Ride    1000
dtype: int64
```

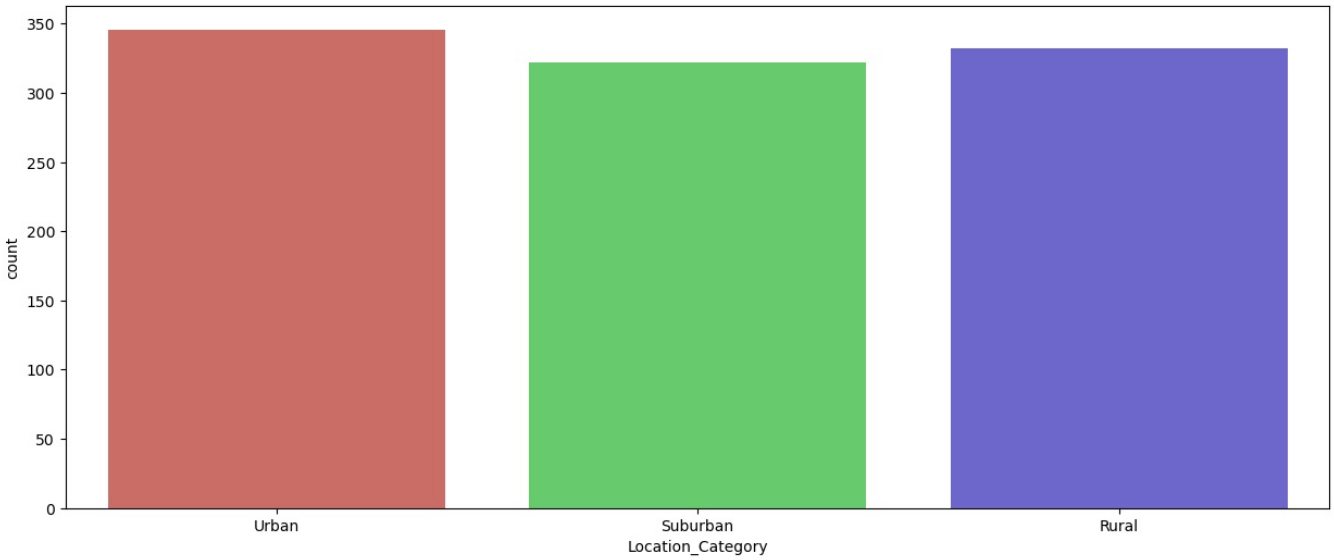
In [12]:
```
df['Location_Category'].unique()
```

Out[12]:
```
array(['Urban', 'Suburban', 'Rural'], dtype=object)
```

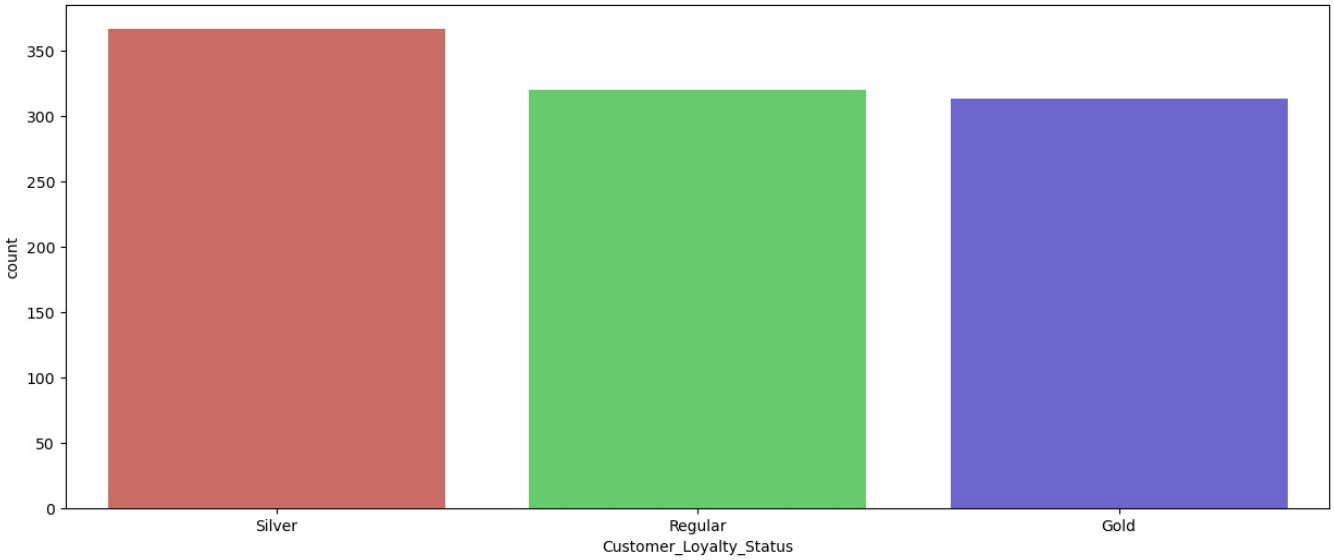
In [13]:
```
df['Location_Category'].value_counts()
```

Out[13]:
```
Location_Category
Urban       346
Rural       332
Suburban    322
Name: count, dtype: int64
```

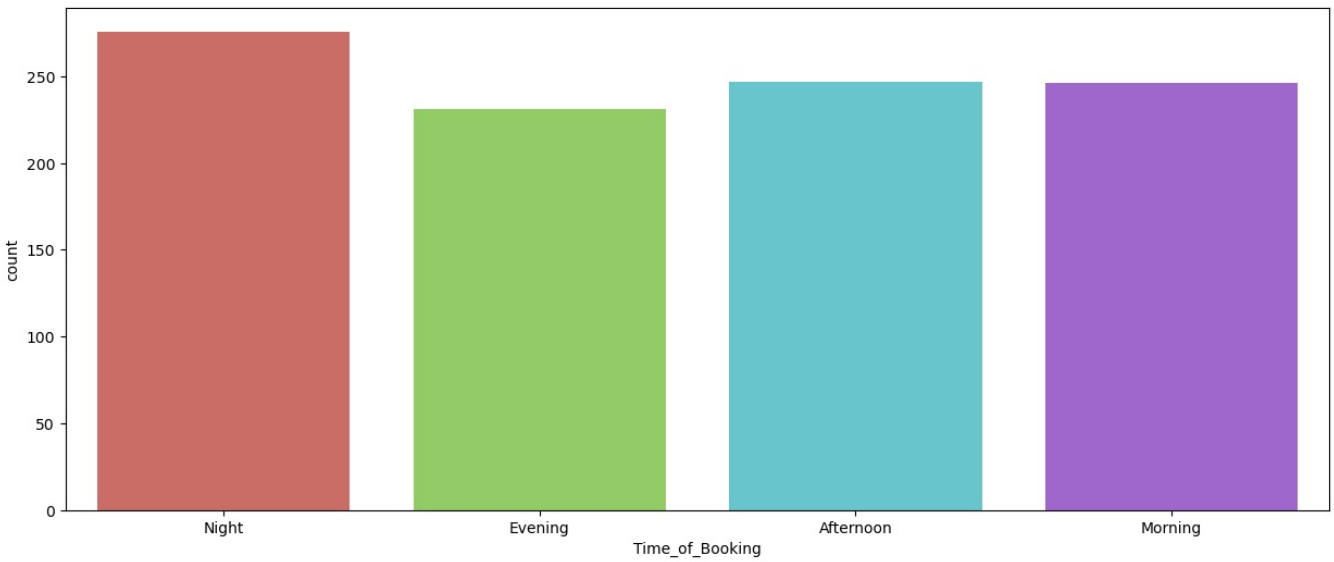
In [16]:
```
plt.figure(figsize=(15, 6))
sns.countplot(x='Location_Category', data=df, palette='hls')
plt.show()
```



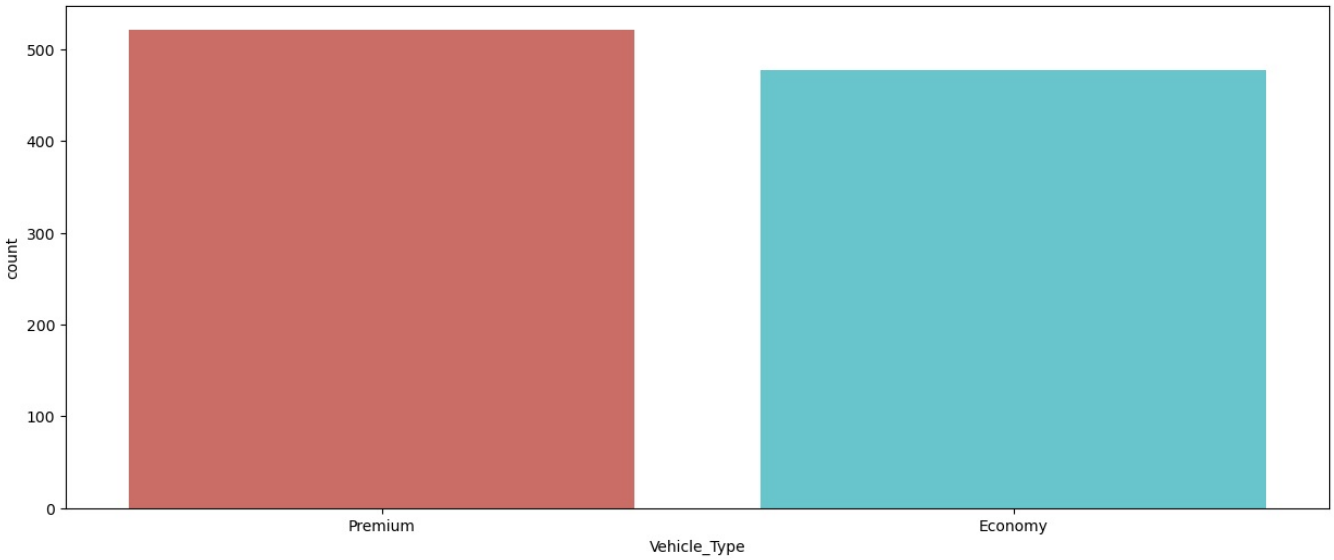
In [17]:
```
plt.figure(figsize=(15, 6))
sns.countplot(x='Customer_Loyalty_Status', data=df, palette='hls')
plt.show()
```



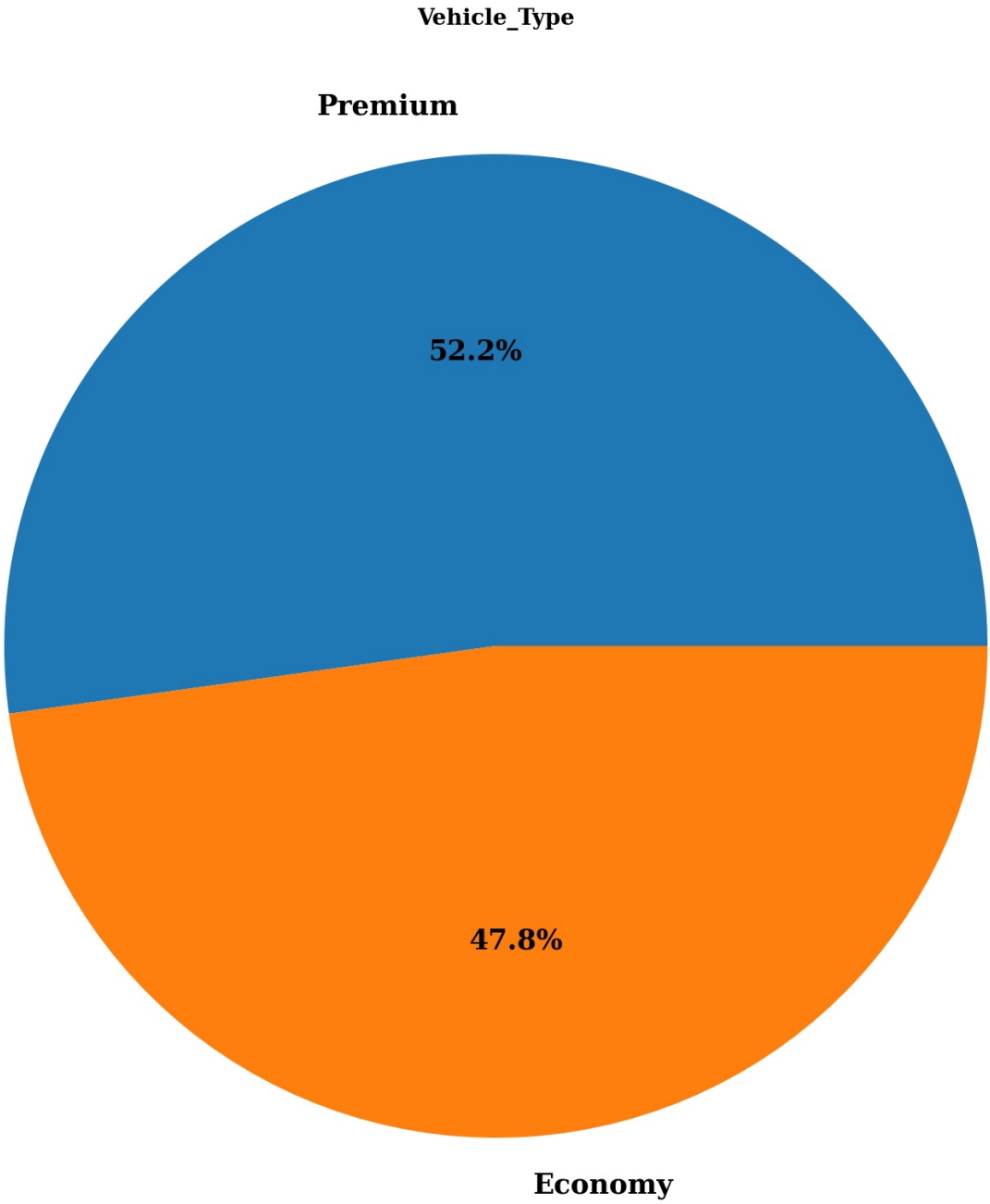
In [18]:
```
plt.figure(figsize=(15, 6))
sns.countplot(x='Time_of_Booking', data=df, palette='hls')
plt.show()
```



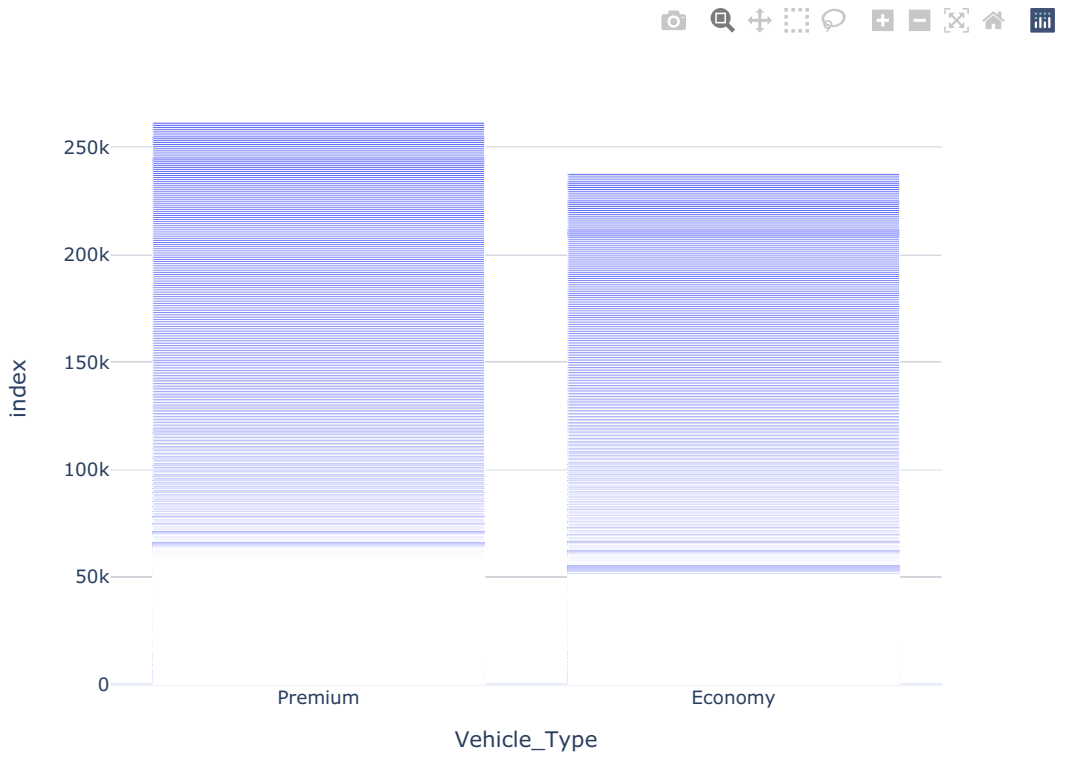
In [19]:
```
plt.figure(figsize=(15, 6))
sns.countplot(x='Vehicle_Type', data=df, palette='hls')
plt.show()
```



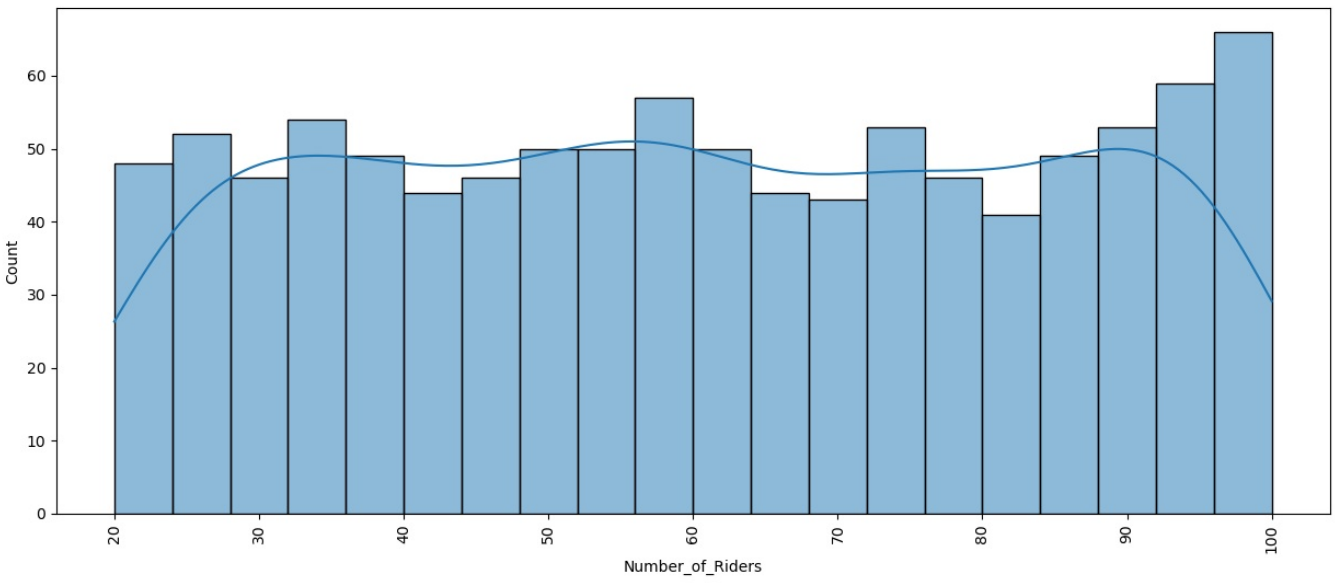
In [20]:
```
plt.figure(figsize=(30,20))
plt.pie(df['Vehicle_Type'].value_counts(), labels=df['Vehicle_Type'].value_counts().index, autopct='%1.1f%%', t
                                           'color': 'black',
                                           'weight': 'bold',
                                           'family': 'serif' })
hfont = {'fontname':'serif', 'weight': 'bold'}
plt.title('Vehicle_Type', size=20, **hfont)
plt.show()
```

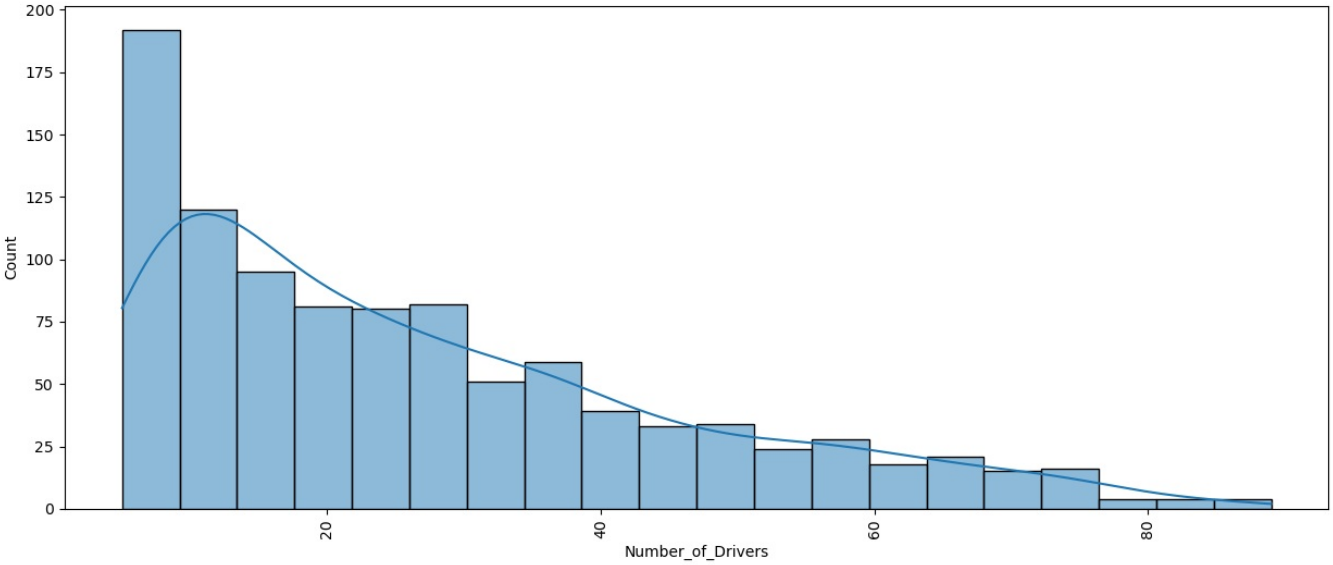


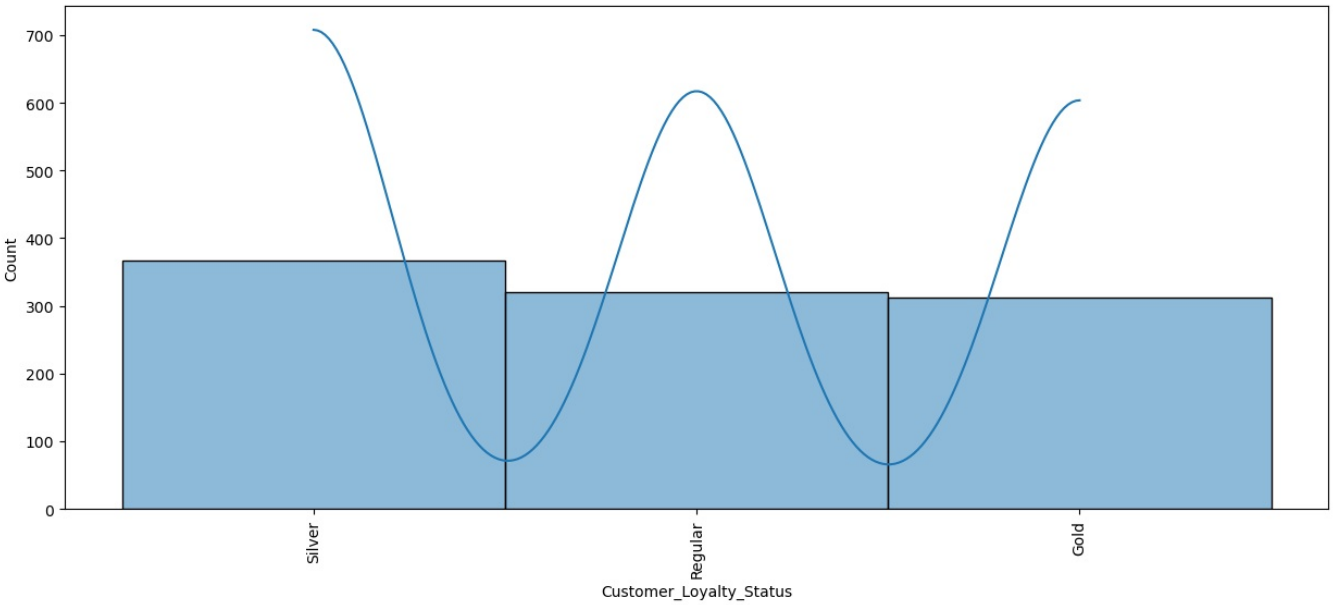
In [21]:
```
import plotly.express as px

fig = px.bar(df, x="Vehicle_Type", y= df.index)
fig.show()
```

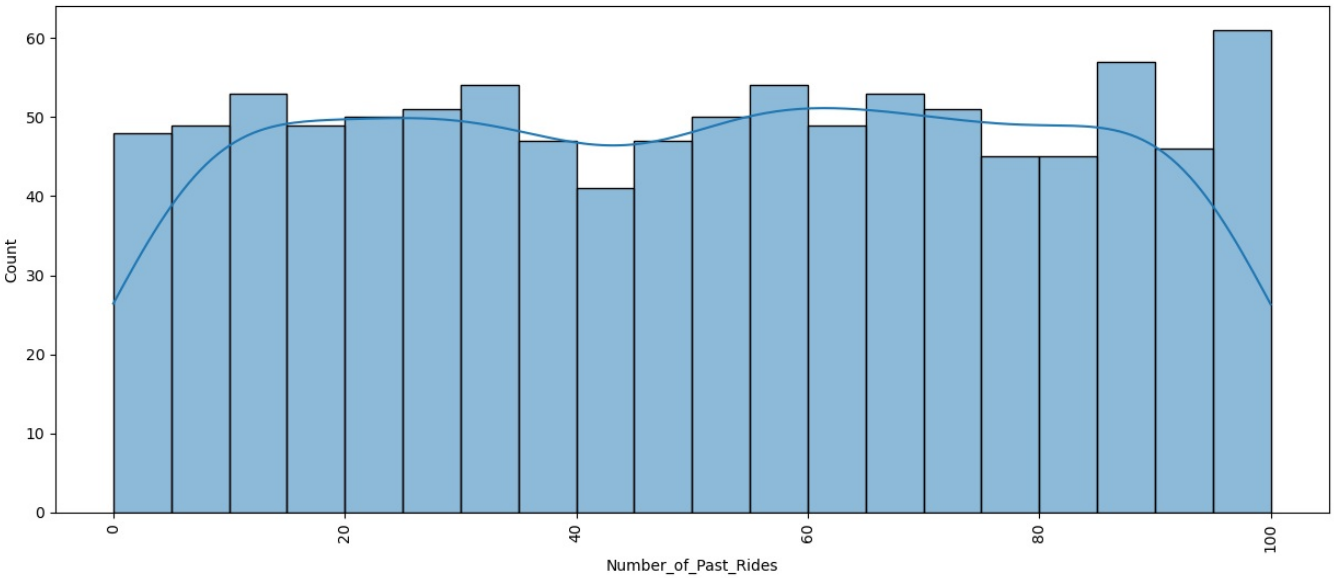


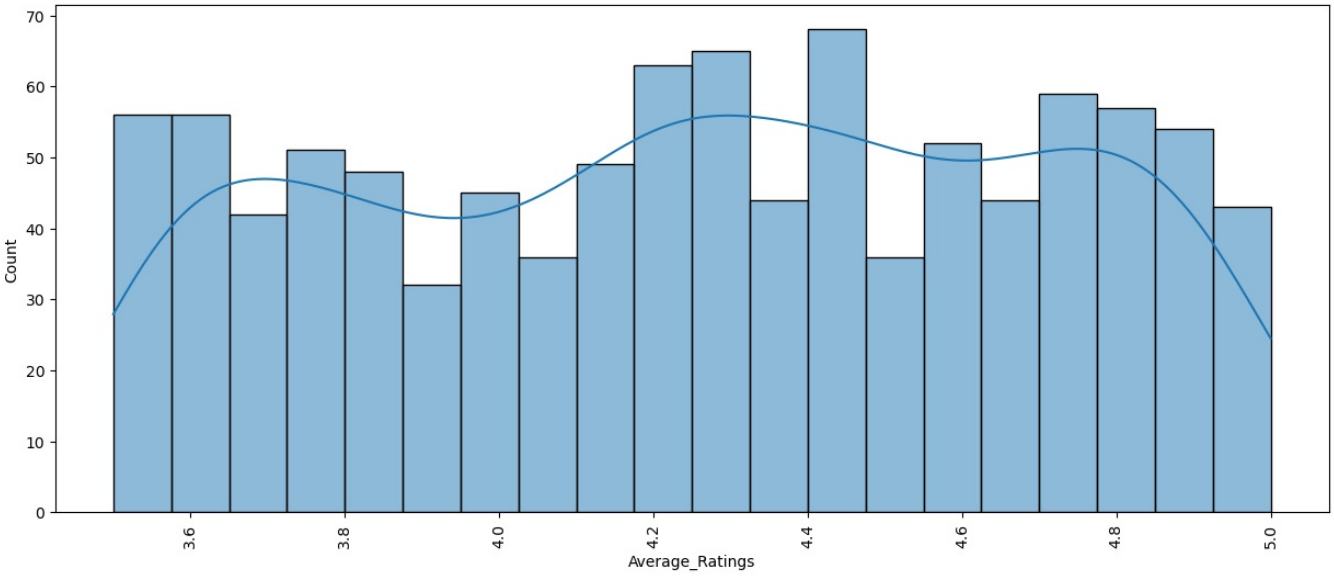
In [22]:
```
for i in df.columns:
    if i != 'Location_Category':
        plt.figure(figsize=(15,6))
        sns.histplot(df[i], kde = True, bins = 20, palette = 'hls')
        plt.xticks(rotation = 90)
        plt.show()
```

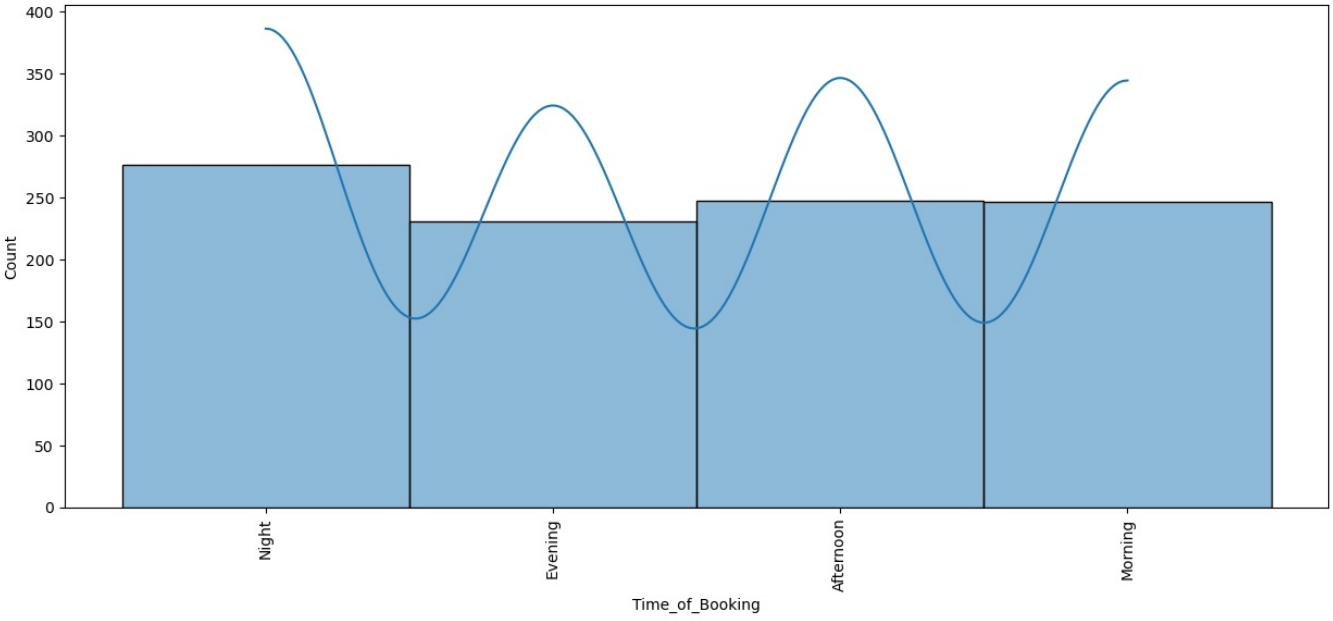


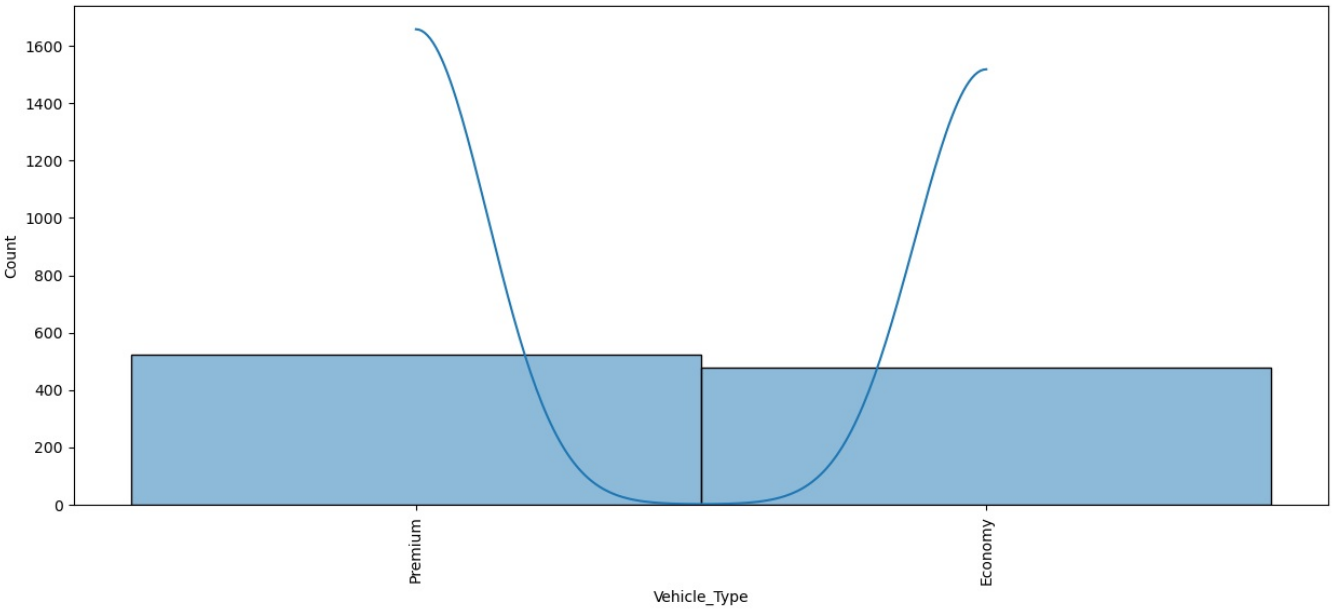

Count
Number_of_Drivers
0
25
50
75
100
125
150
175
200
20
40
60
80

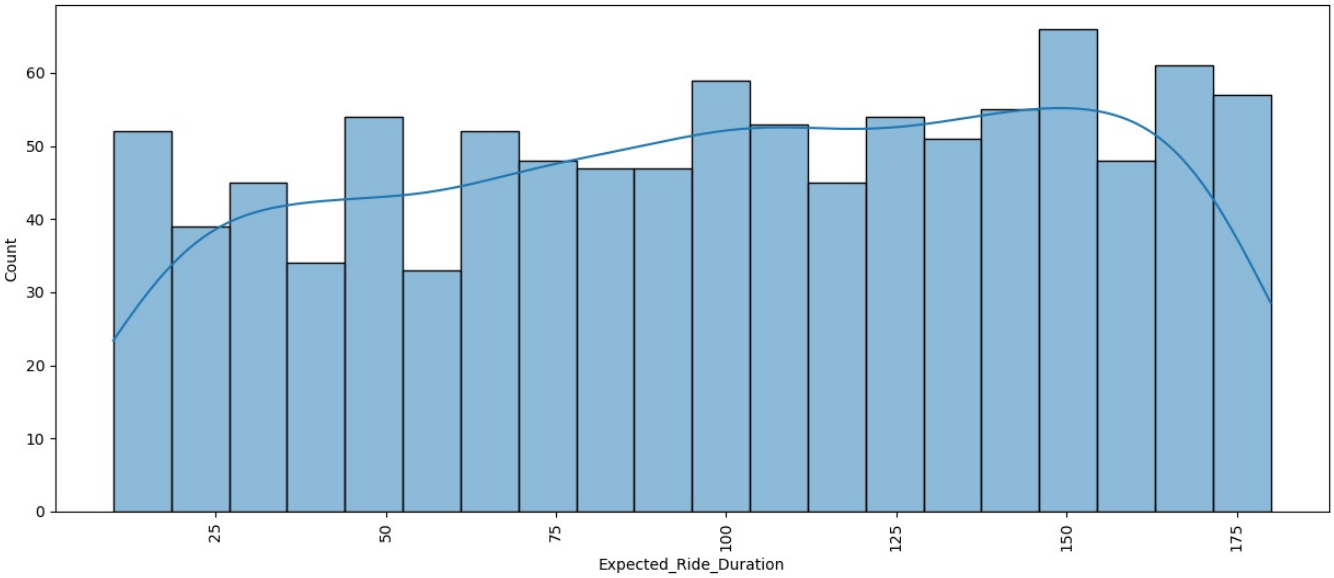

Count
Customer_Loyalty_Status
0
100
200
300
400
500
600
700
Silver
Regular
Gold

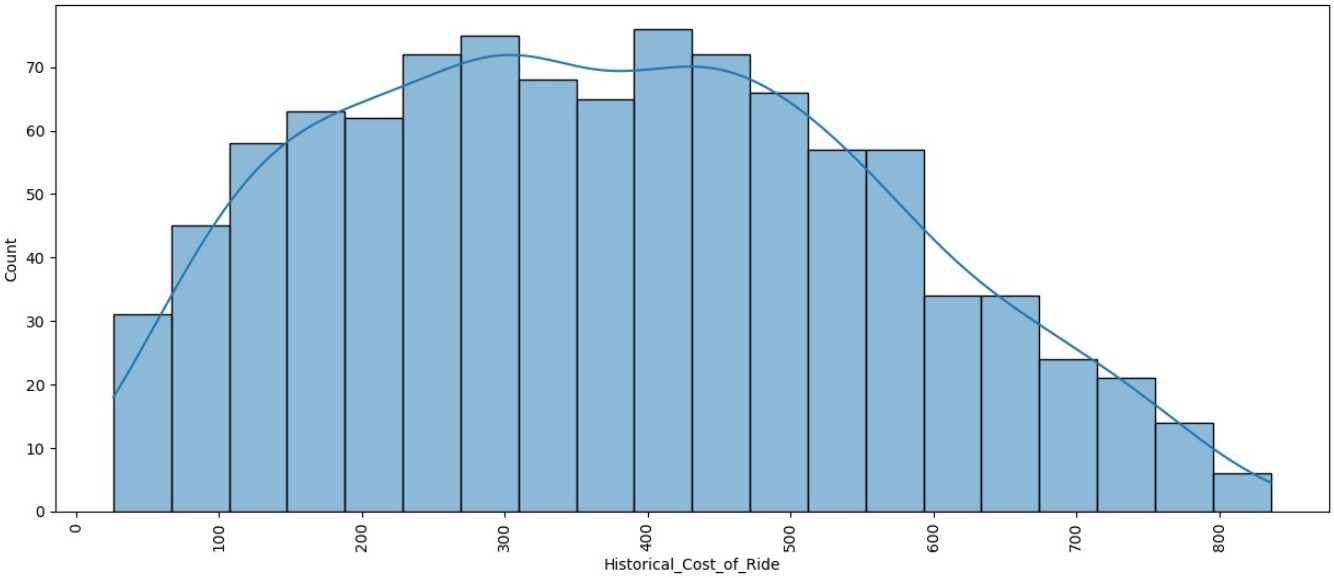

Count
Number_of_Past_Rides
0
20
40
60
80
100
0
10
20
30
40
50
60


Count
Average_Ratings
3.6
3.8
4.0
4.2
4.4
4.6
4.8
5.0
0
10
20
30
40
50
60
70

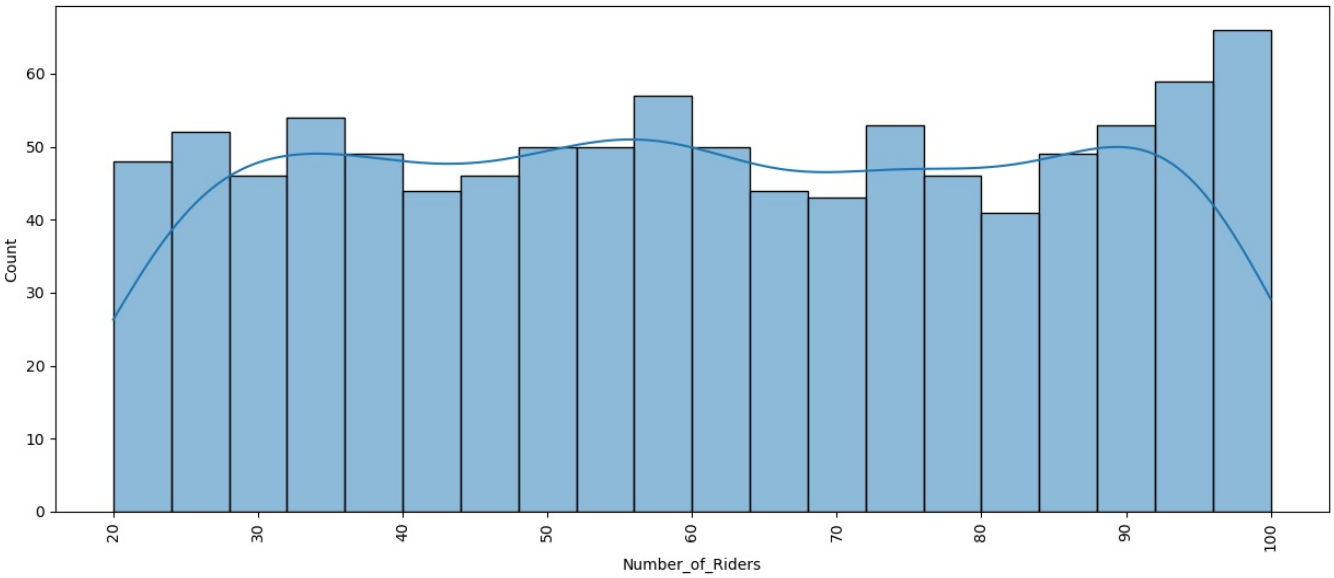

400
350
300
250
200
150
100
50
0
Count
Night
Evening
Afternoon
Morning
Time_of_Booking

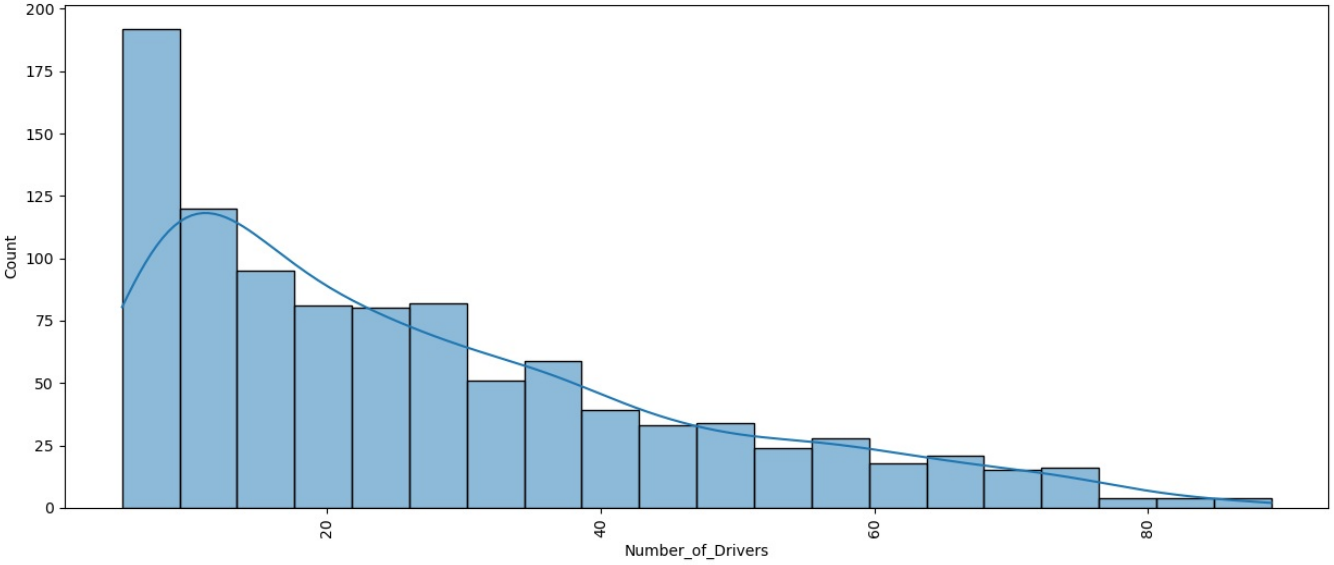

1600
1400
1200
1000
800
600
400
200
0
Count
Premium
Economy
Vehicle_Type

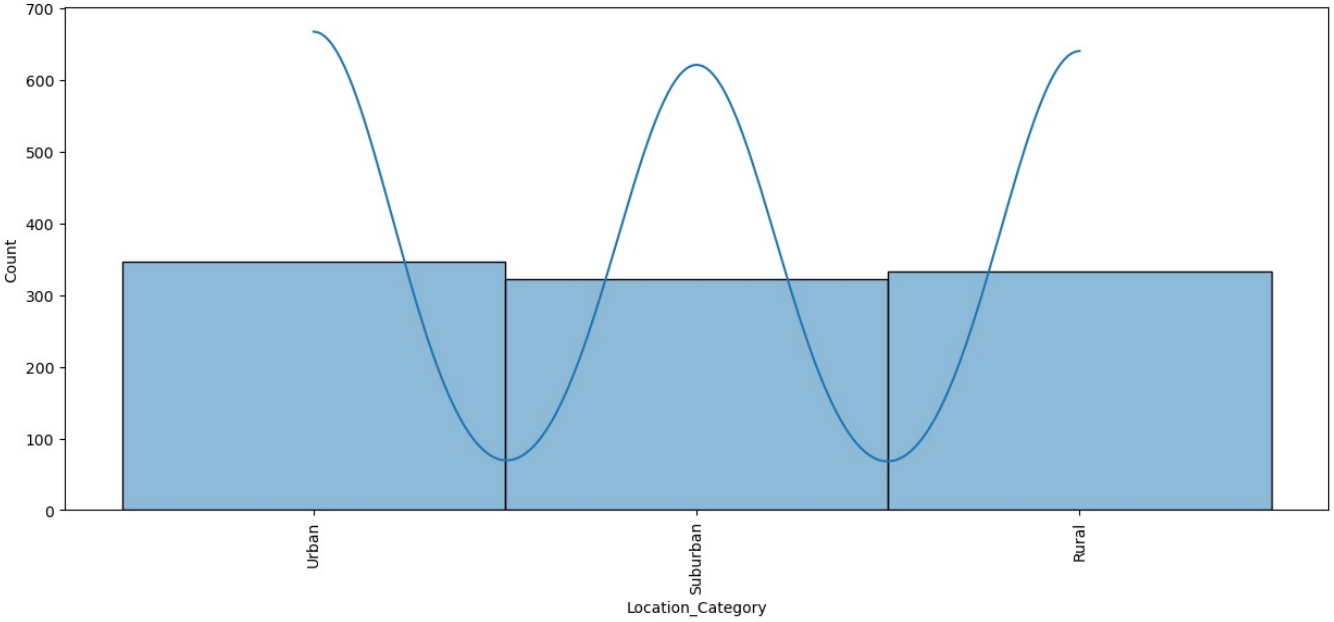


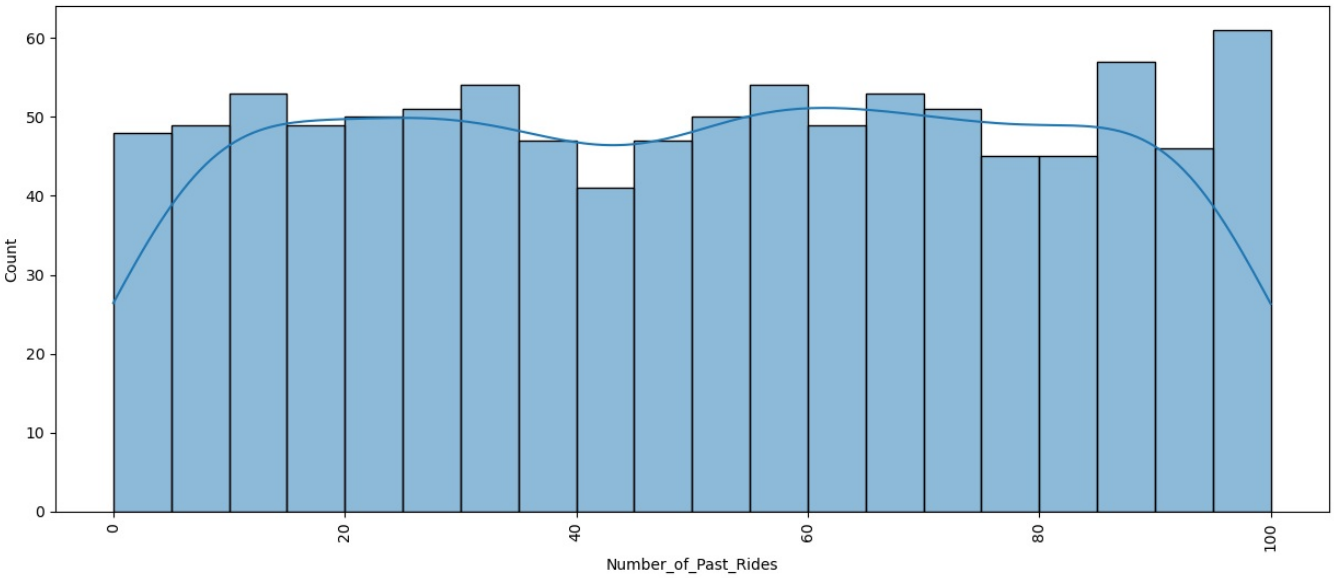


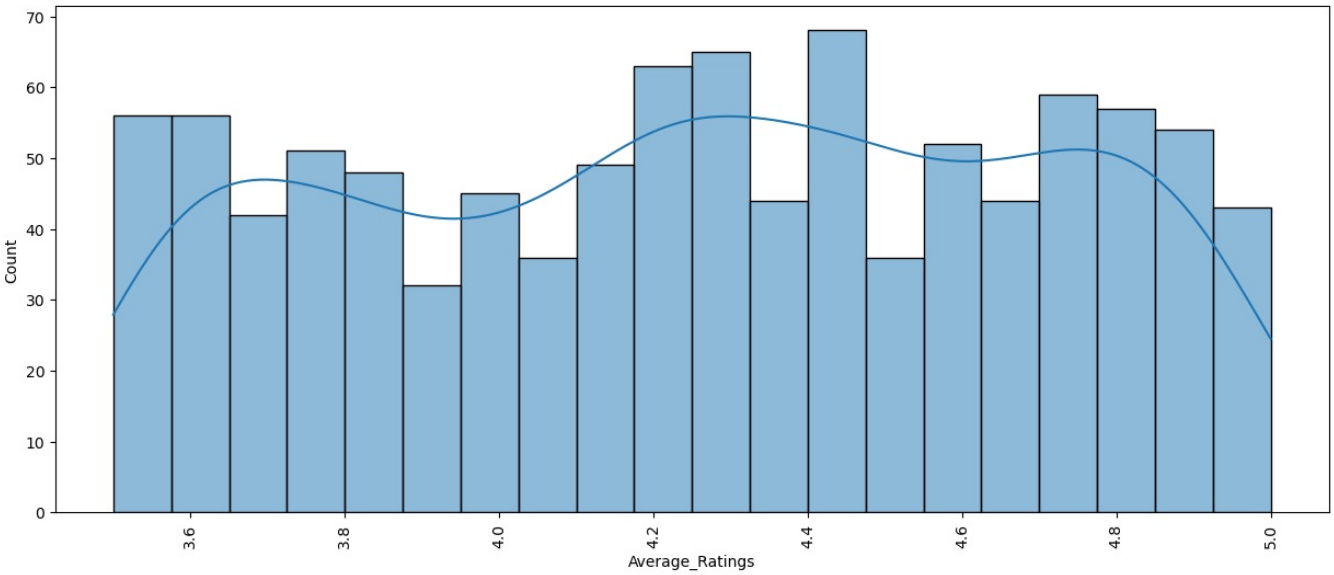
In [23]:
```
for i in df.columns:
    if i != 'Customer_Loyalty_Status':
        plt.figure(figsize=(15,6))
        sns.histplot(df[i], kde = True, bins = 20, palette = 'hls')
        plt.xticks(rotation = 90)
        plt.show()
```

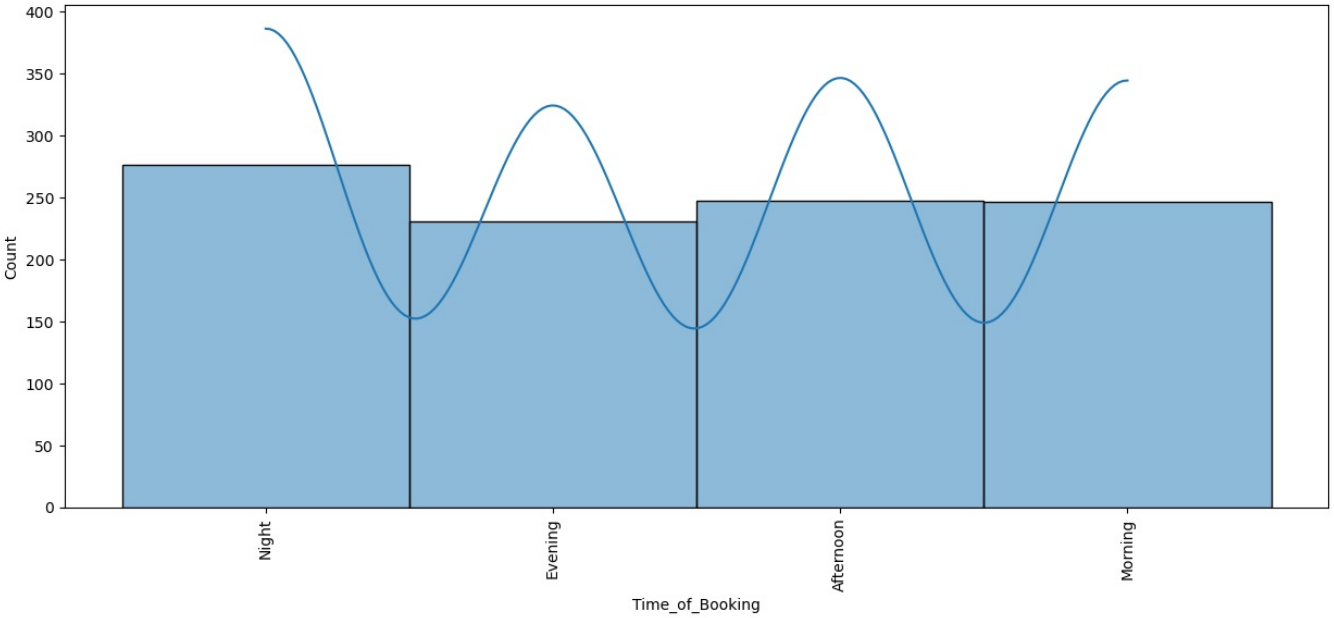

Count
Number_of_Riders

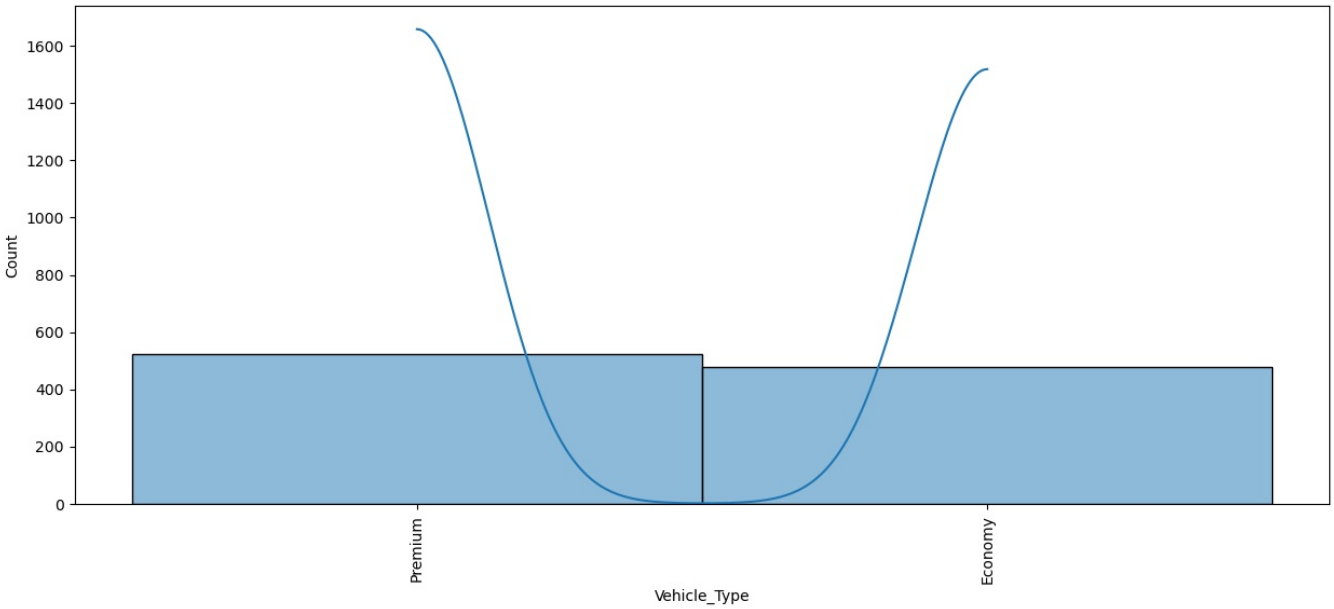

Count
Number_of_Drivers

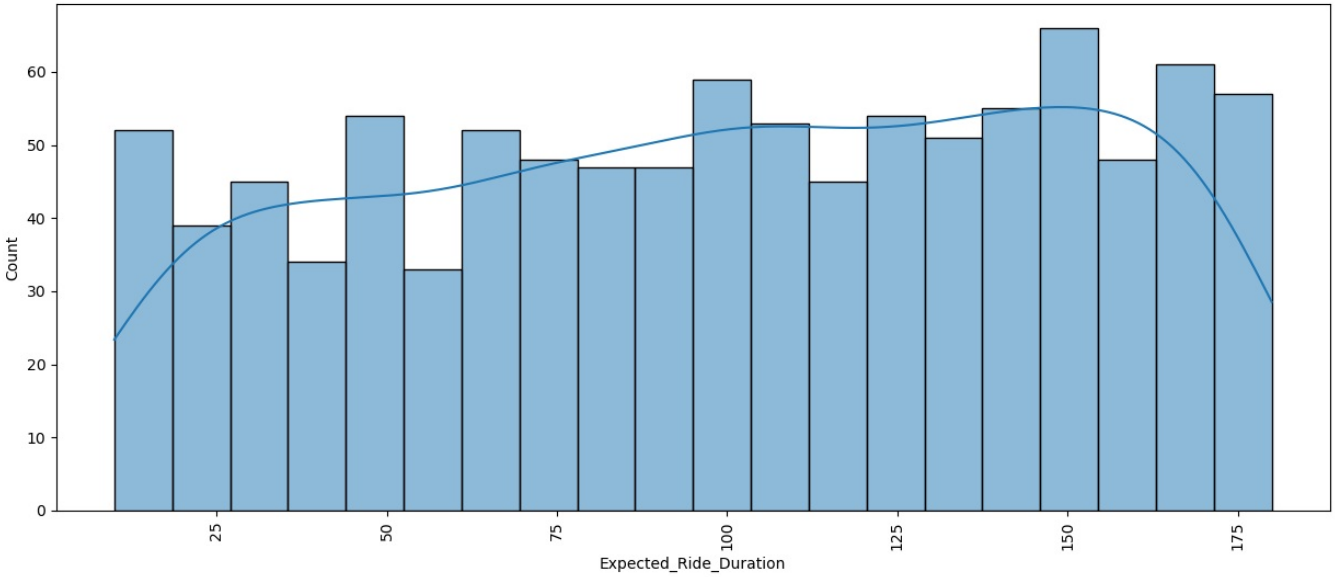

Count
Urban
Suburban
Rural
Location_Category

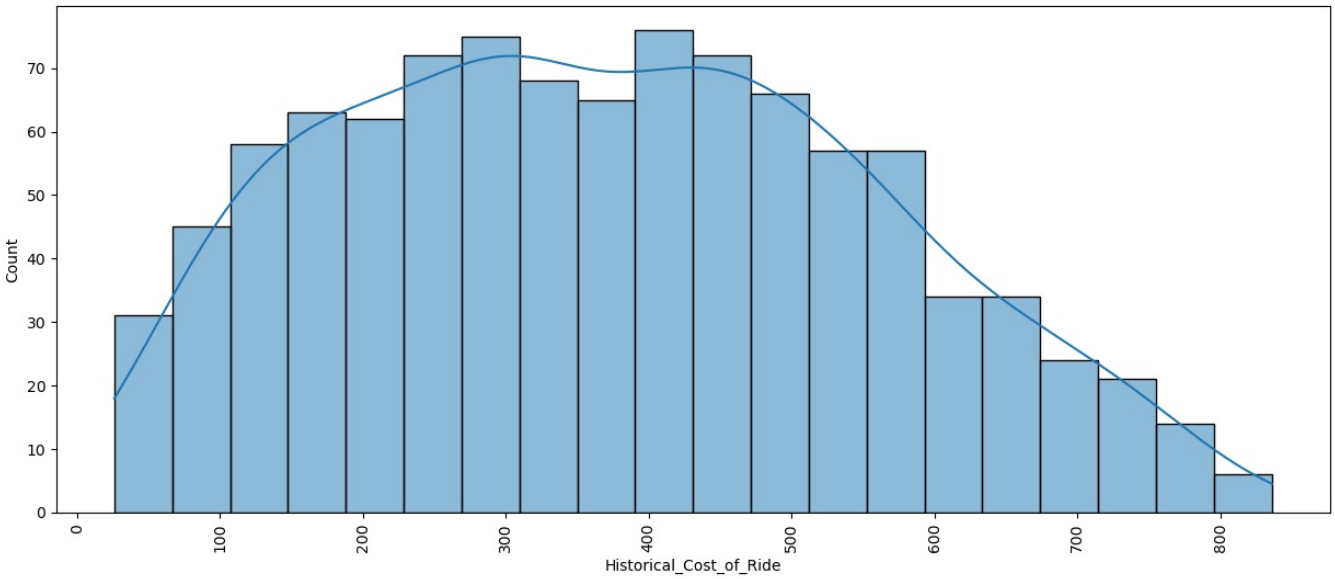

Count
Number_of_Past_Rides


Count
Average_Ratings

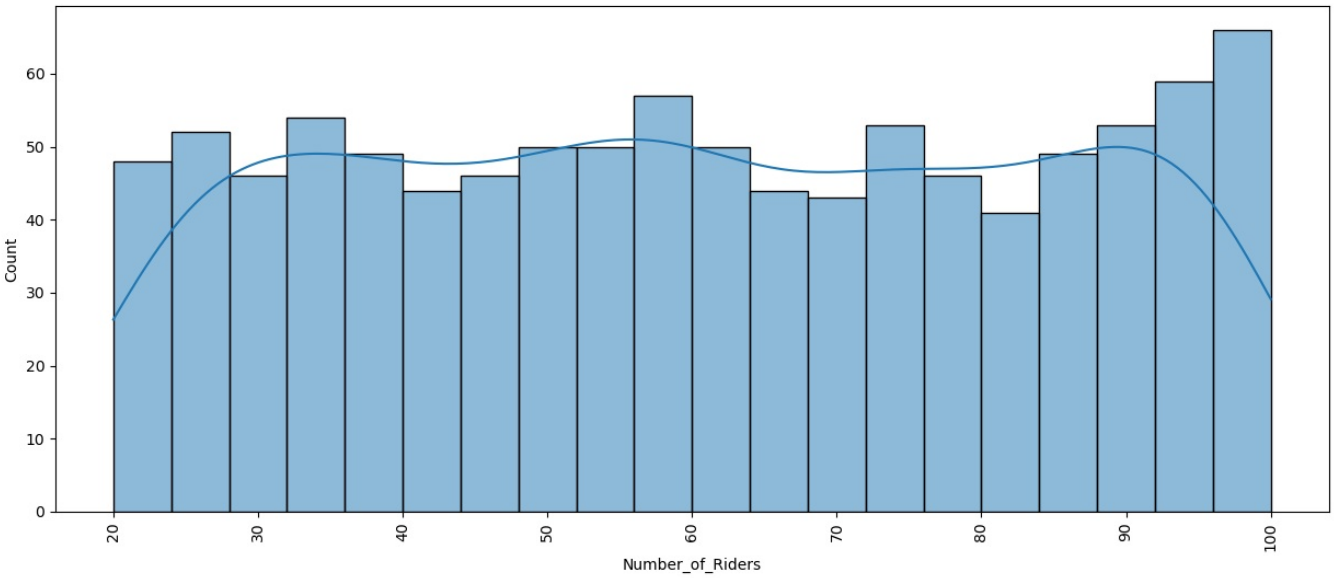

Count
Night
Evening
Afternoon
Morning
Time_of_Booking

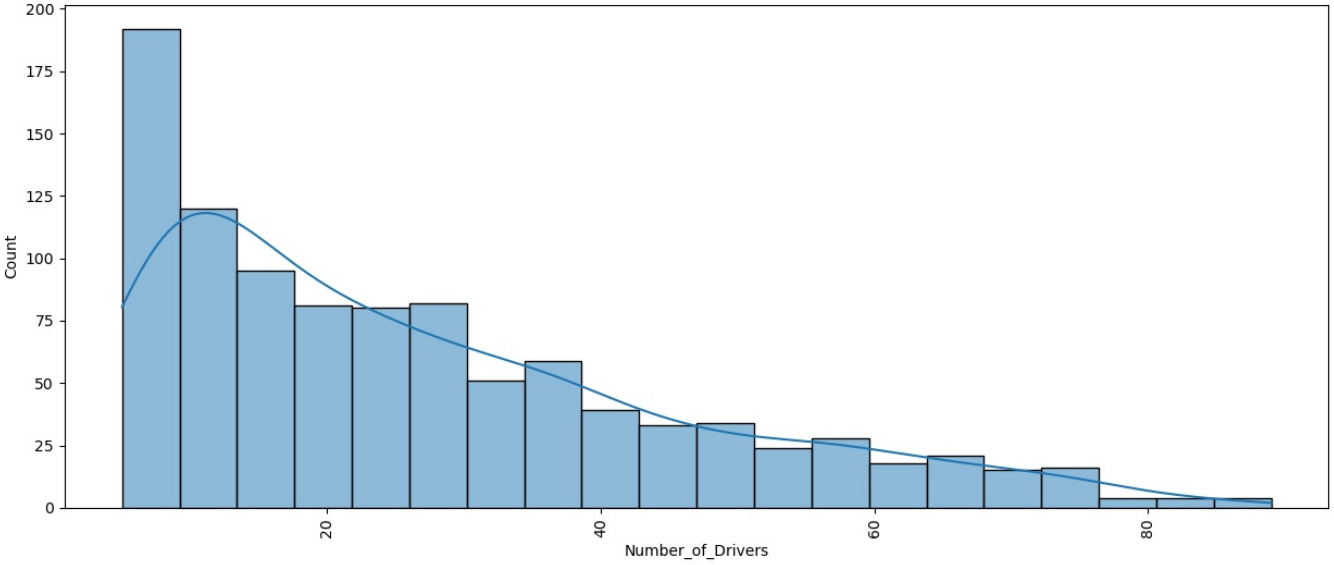


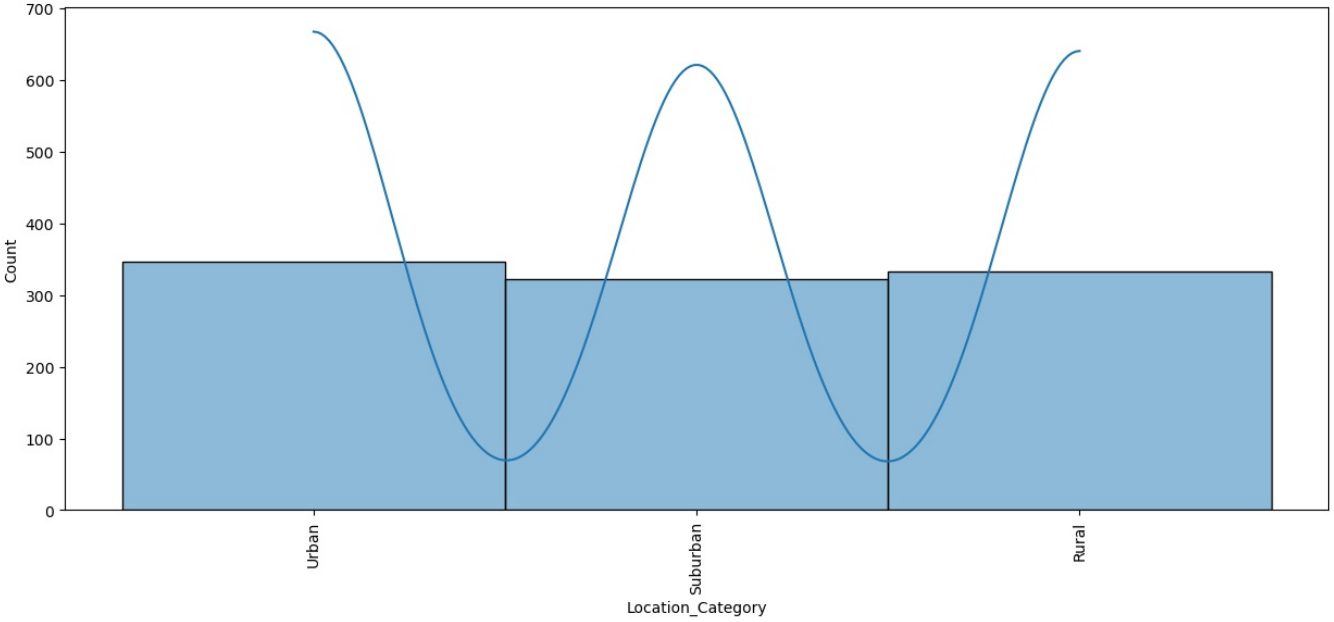


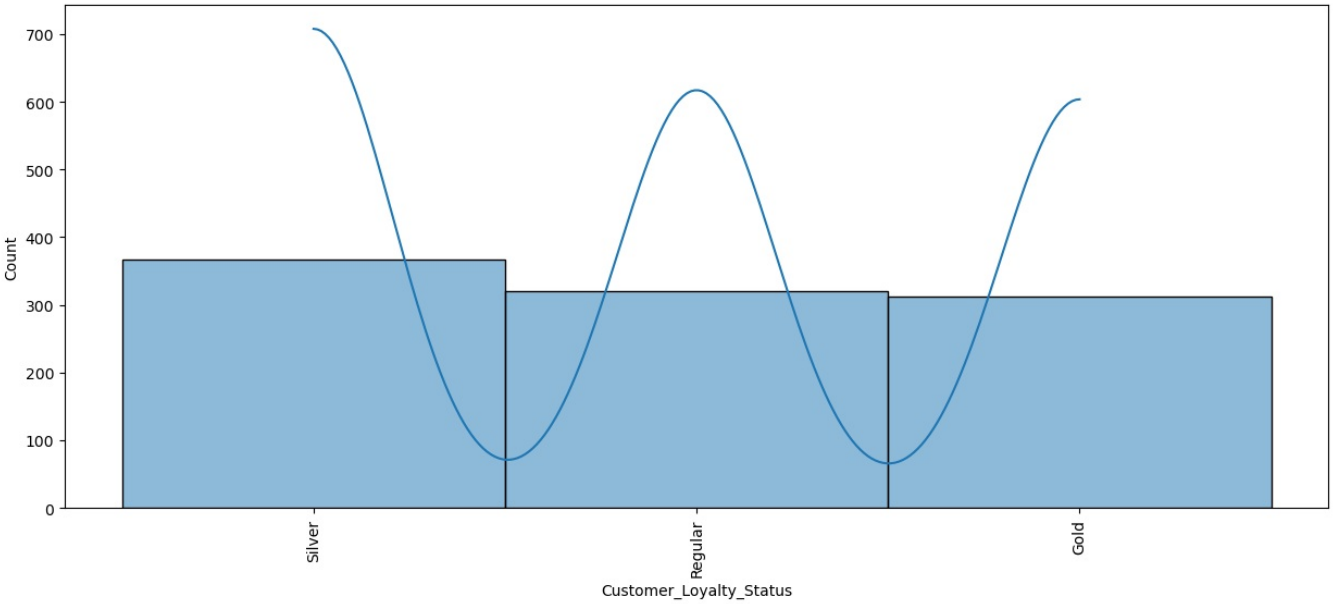


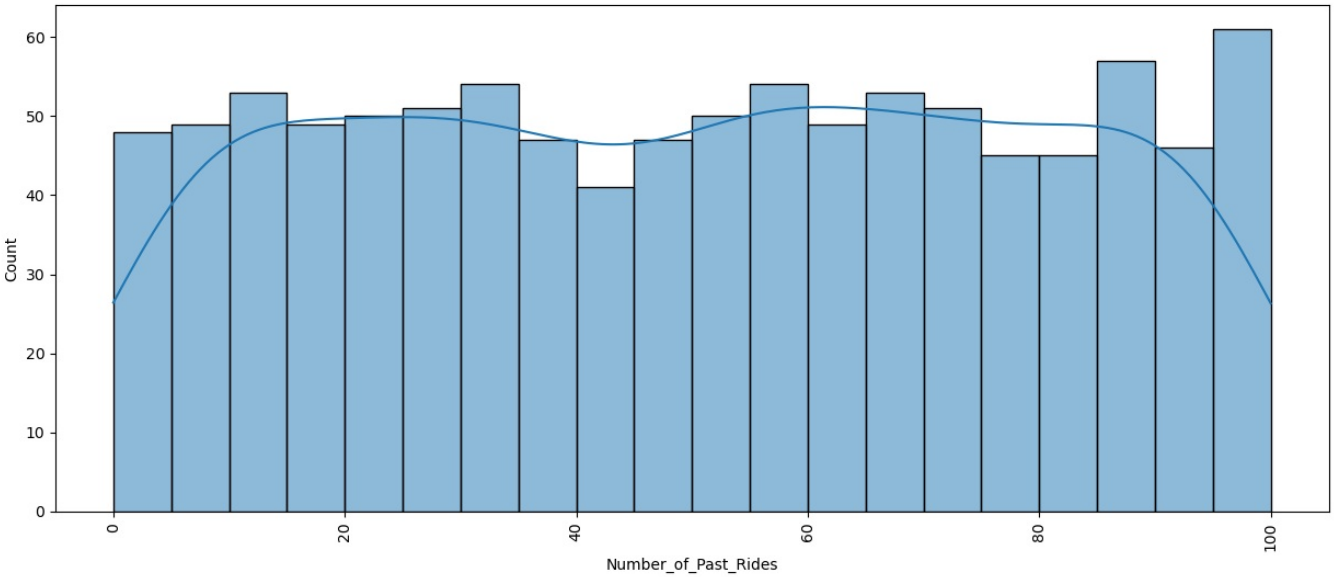
In [24]:
```
for i in df.columns:
    if i != 'Time_of_Booking':
        plt.figure(figsize=(15,6))
        sns.histplot(df[i], kde = True, bins = 20, palette = 'hls')
        plt.xticks(rotation = 90)
        plt.show()
```

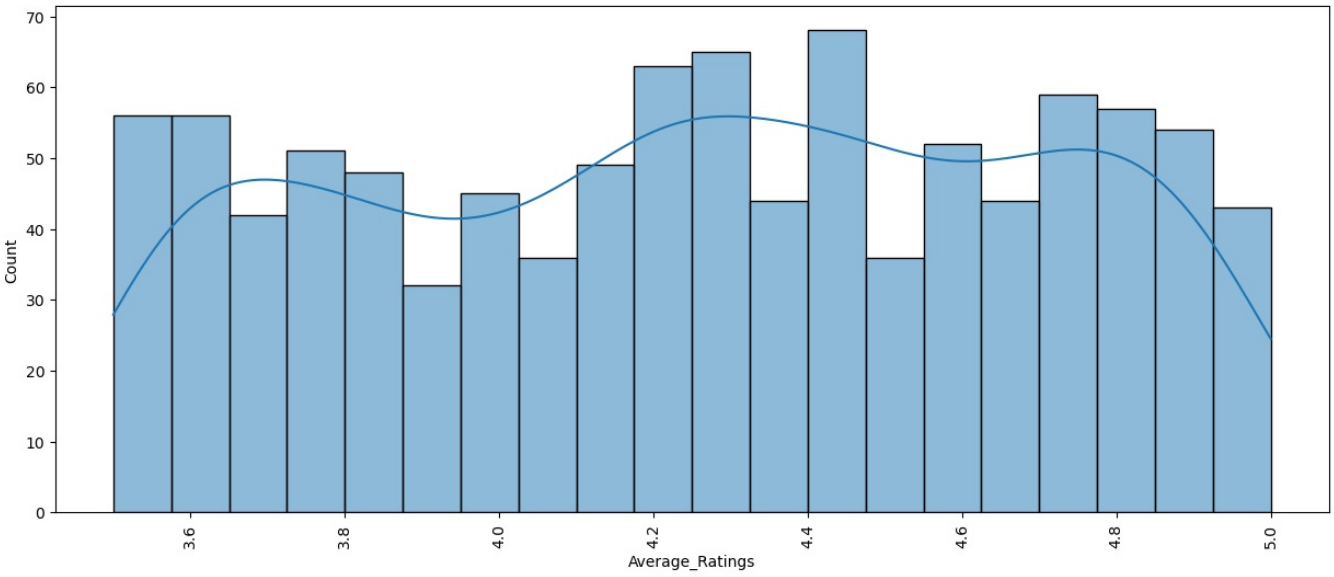

Count
Number_of_Riders

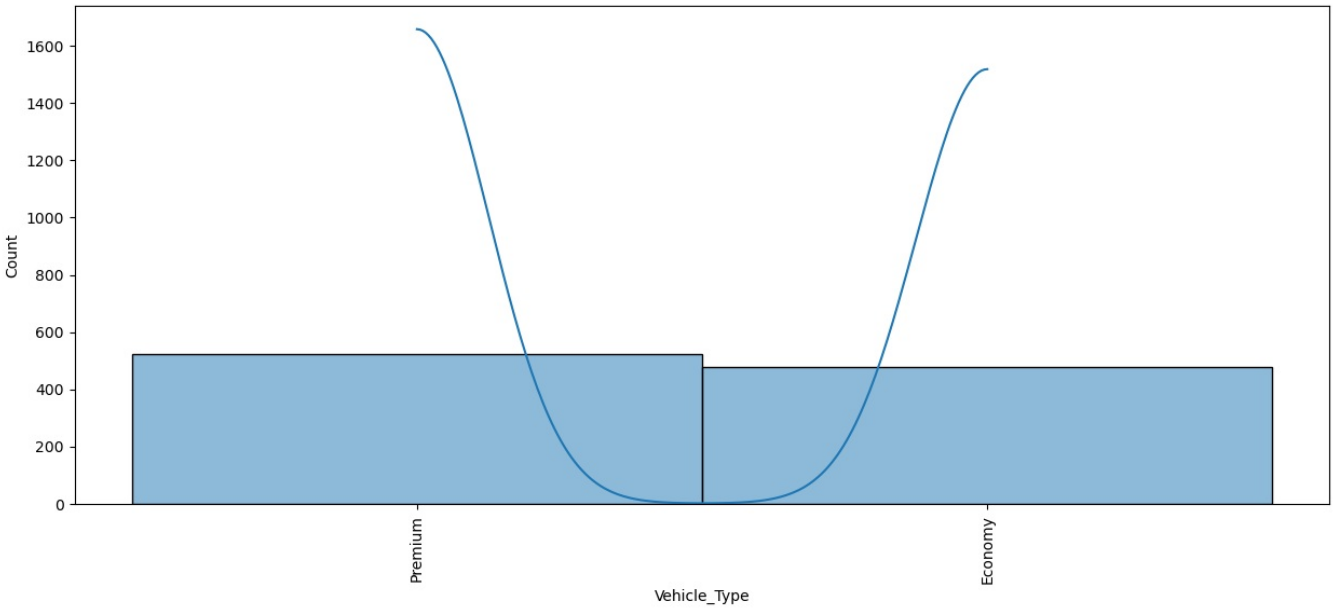

Count
Number_of_Drivers

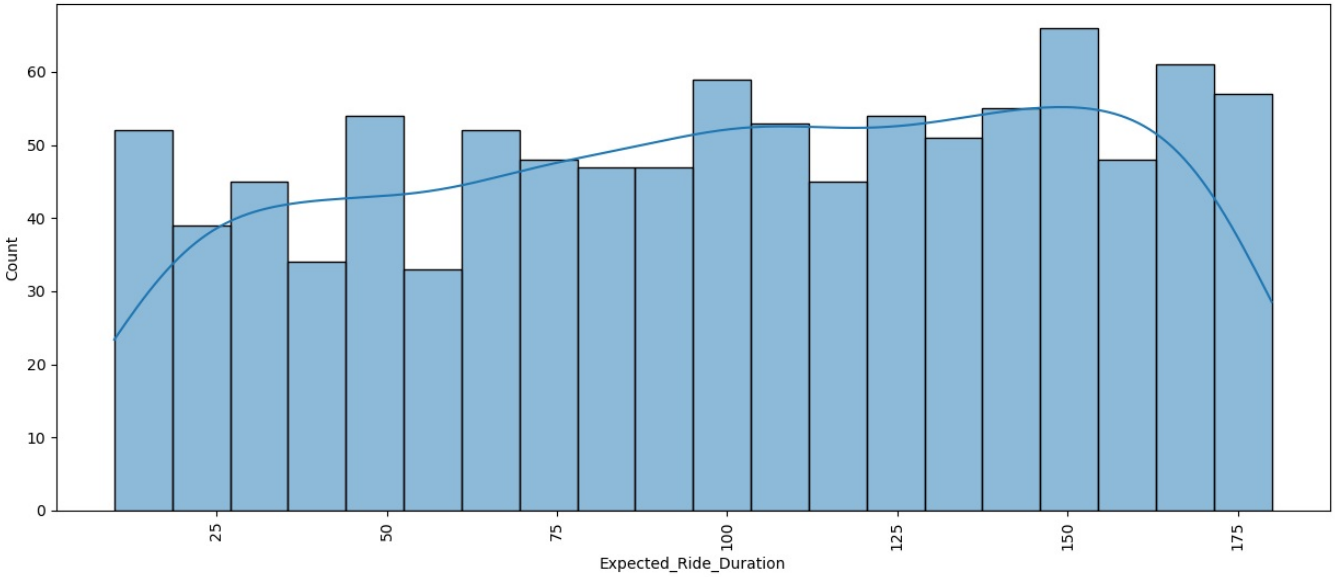

Count
Urban
Suburban
Rural
Location_Category

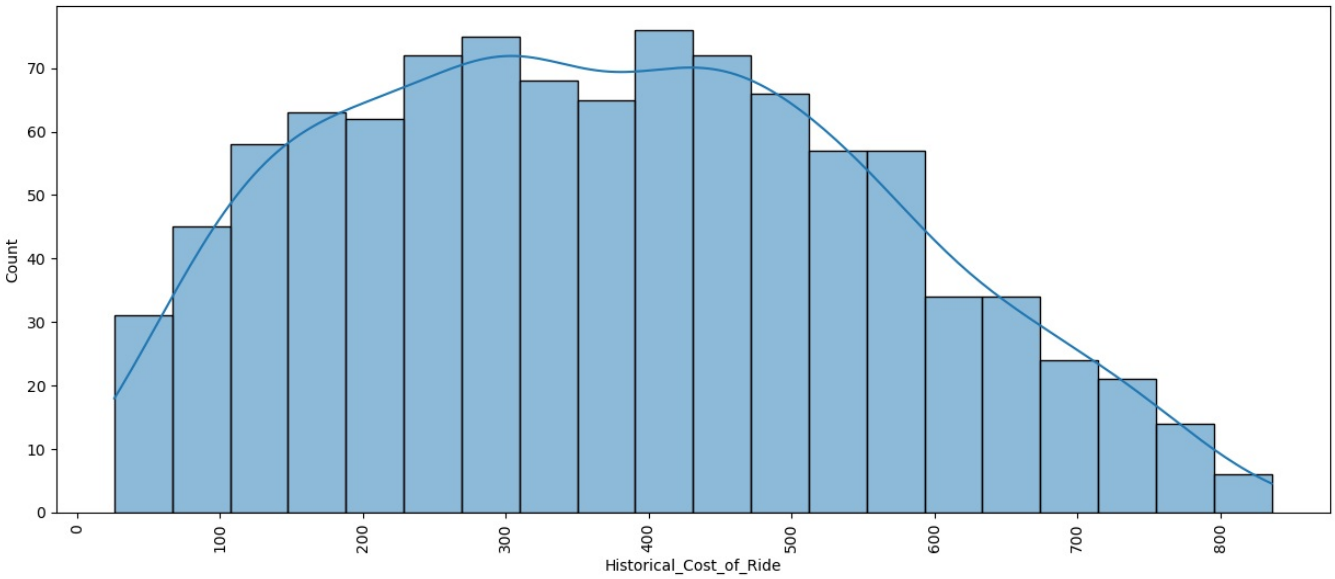

Count
Silver
Regular
Gold
Customer_Loyalty_Status


Count
Number_of_Past_Rides

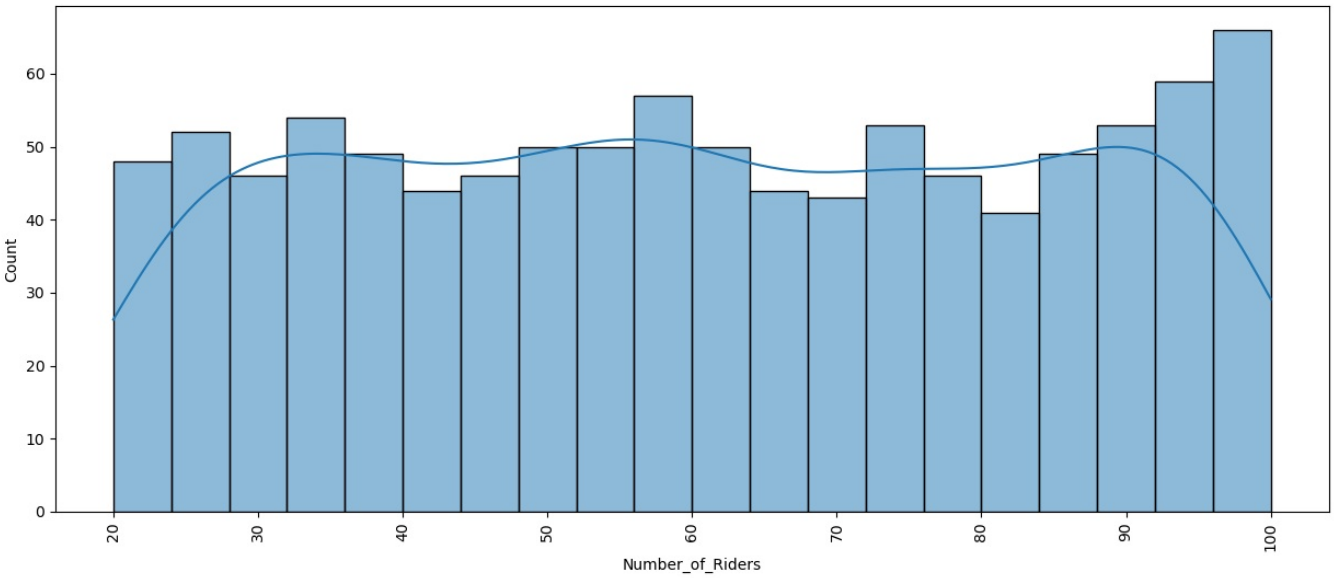

Count
Average_Ratings

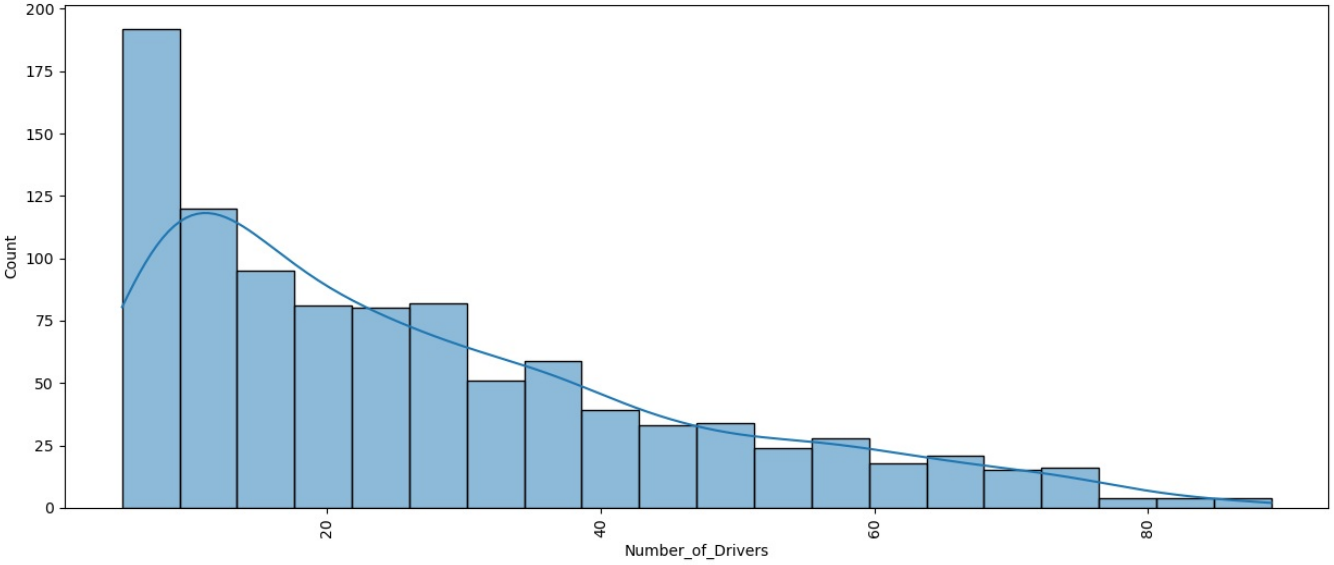


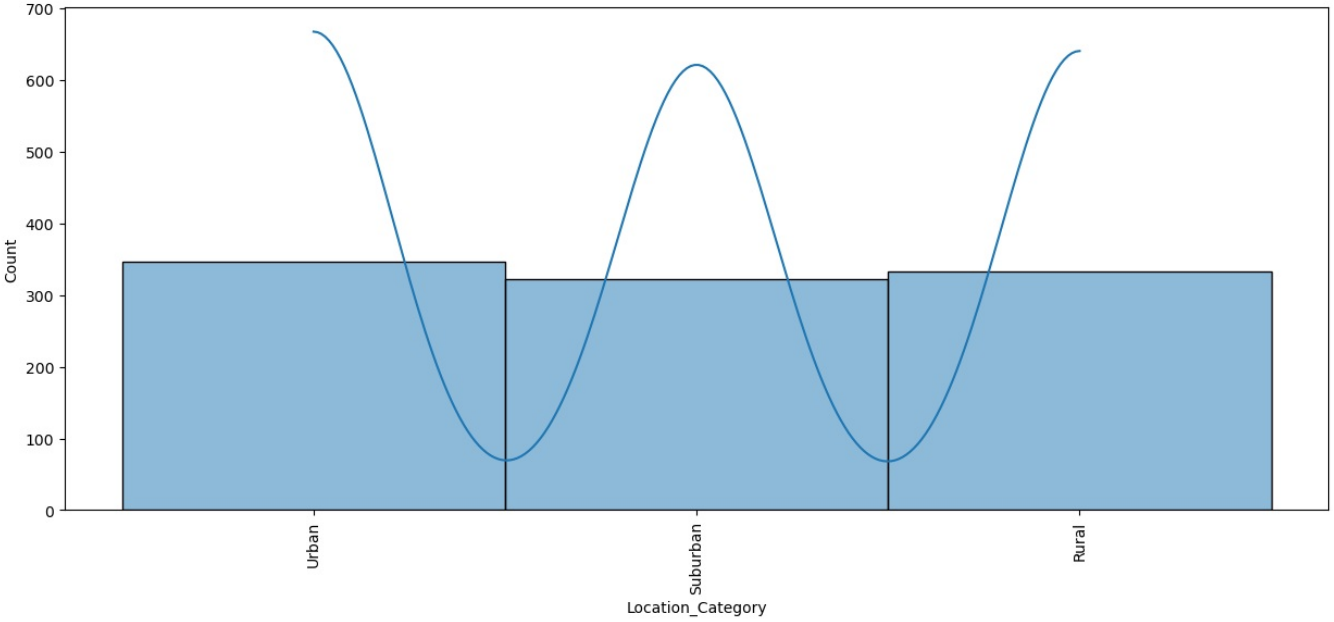


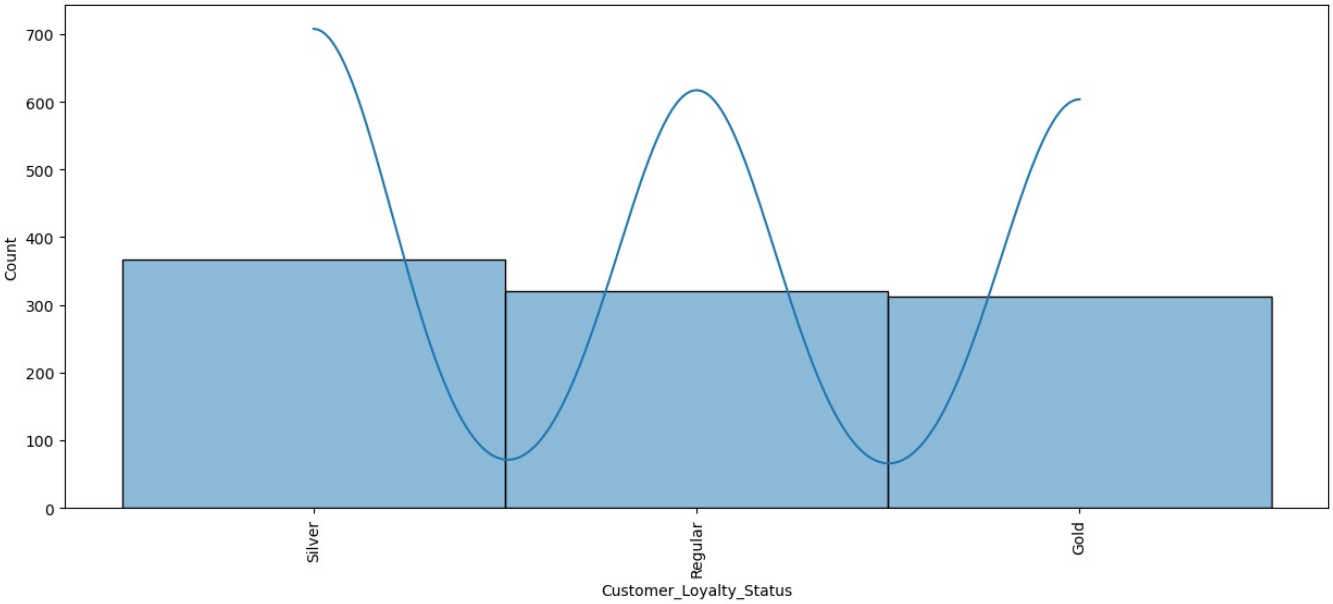


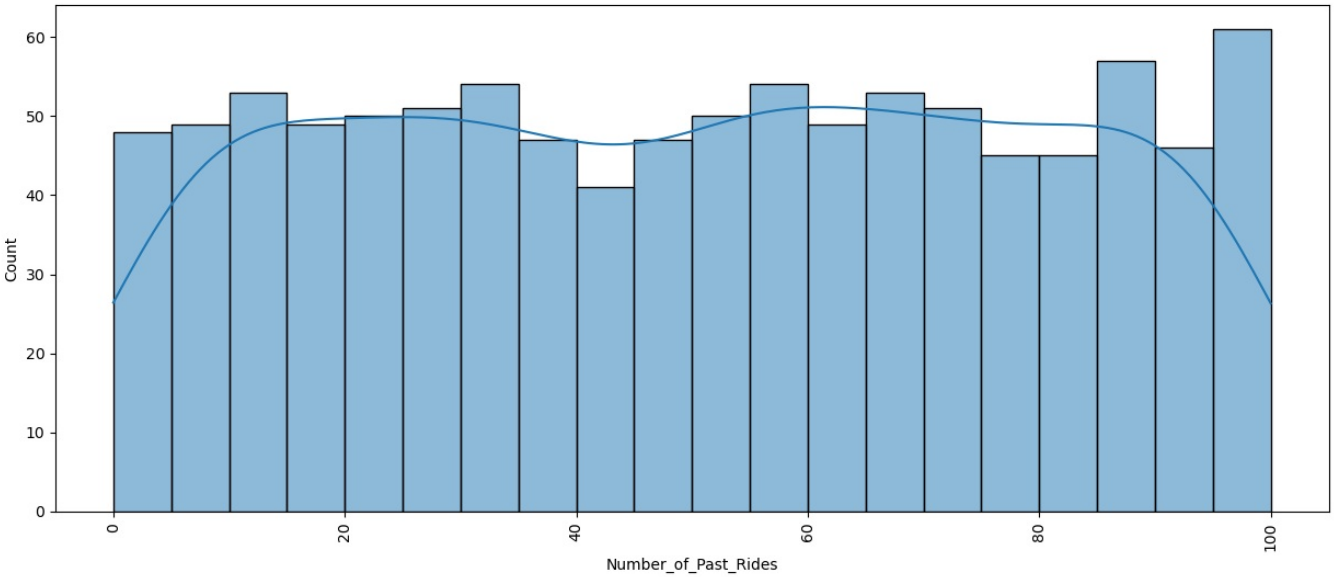
In [25]:
```
for i in df.columns:
    if i != 'Vehicle_Type':
        plt.figure(figsize=(15,6))
        sns.histplot(df[i], kde = True, bins = 20, palette = 'hls')
        plt.xticks(rotation = 90)
        plt.show()
```

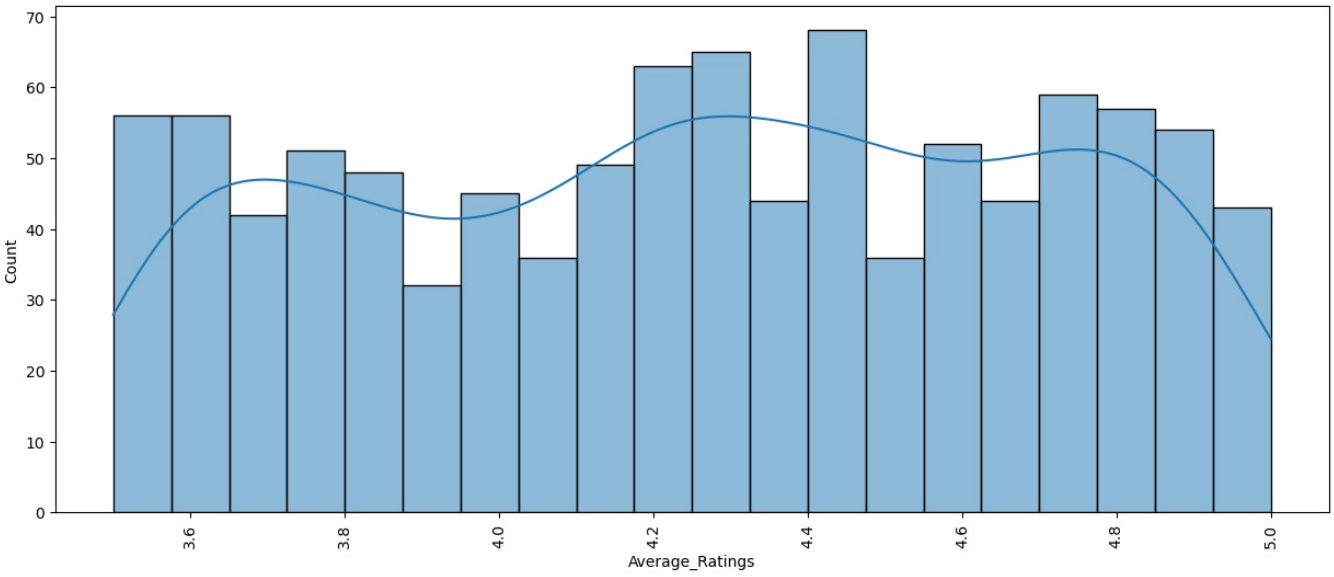

Count
Number_of_Riders

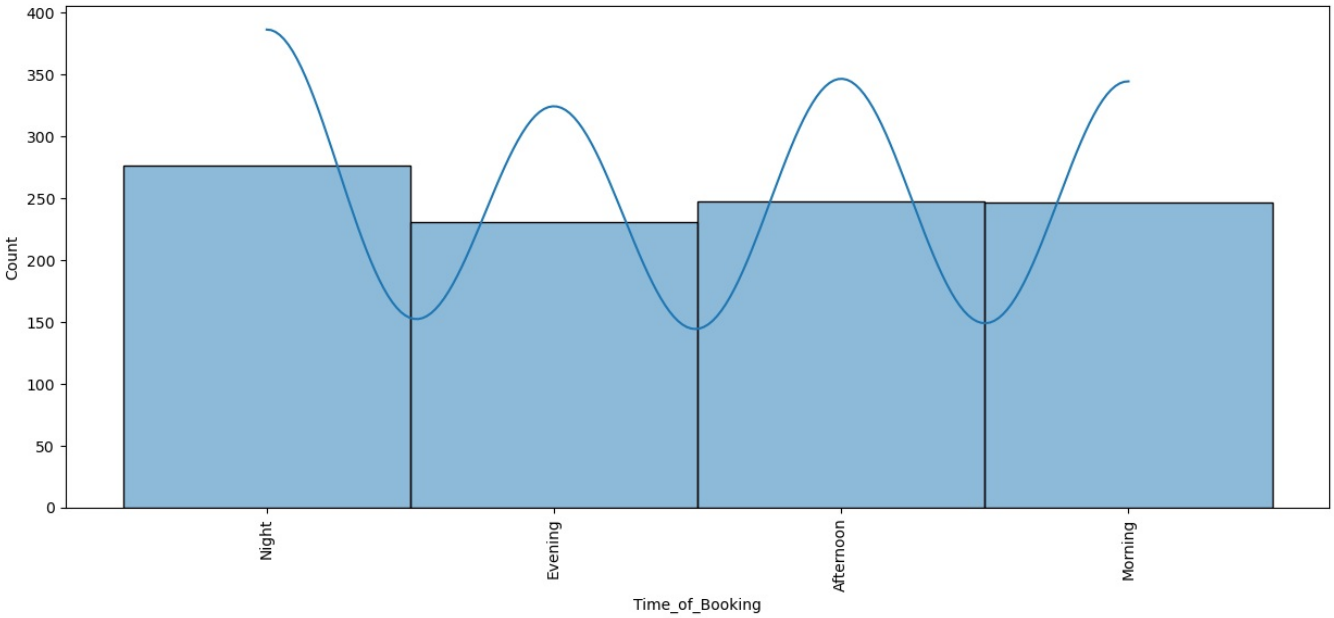

Count
Number_of_Drivers

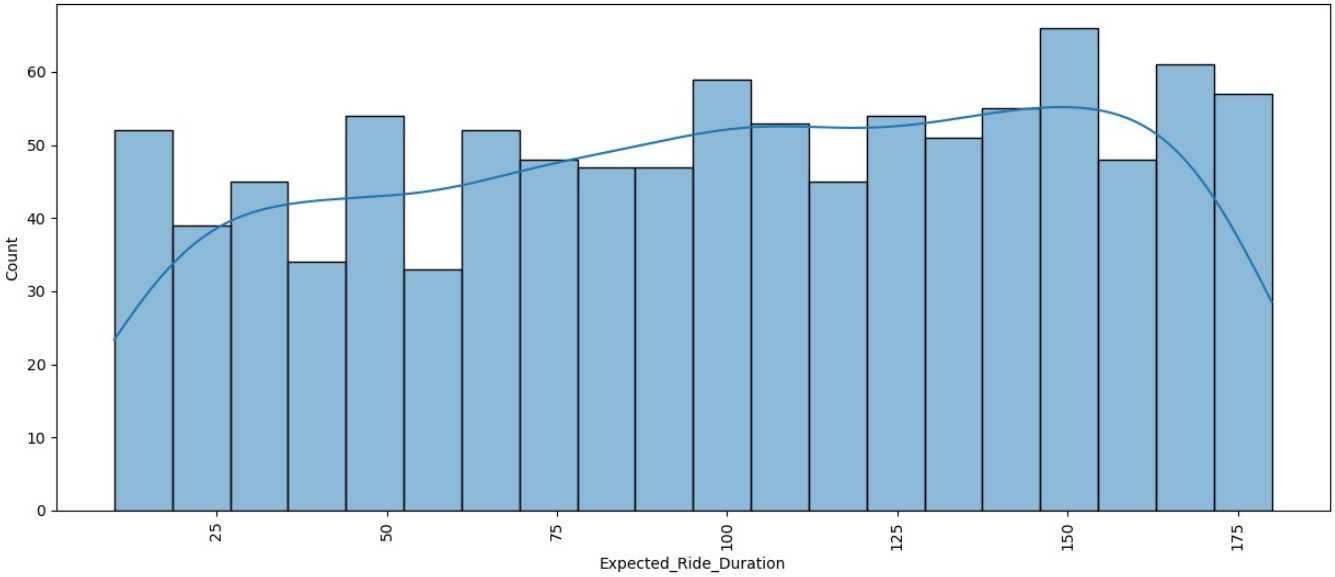

Count
Urban
Suburban
Rural
Location_Category

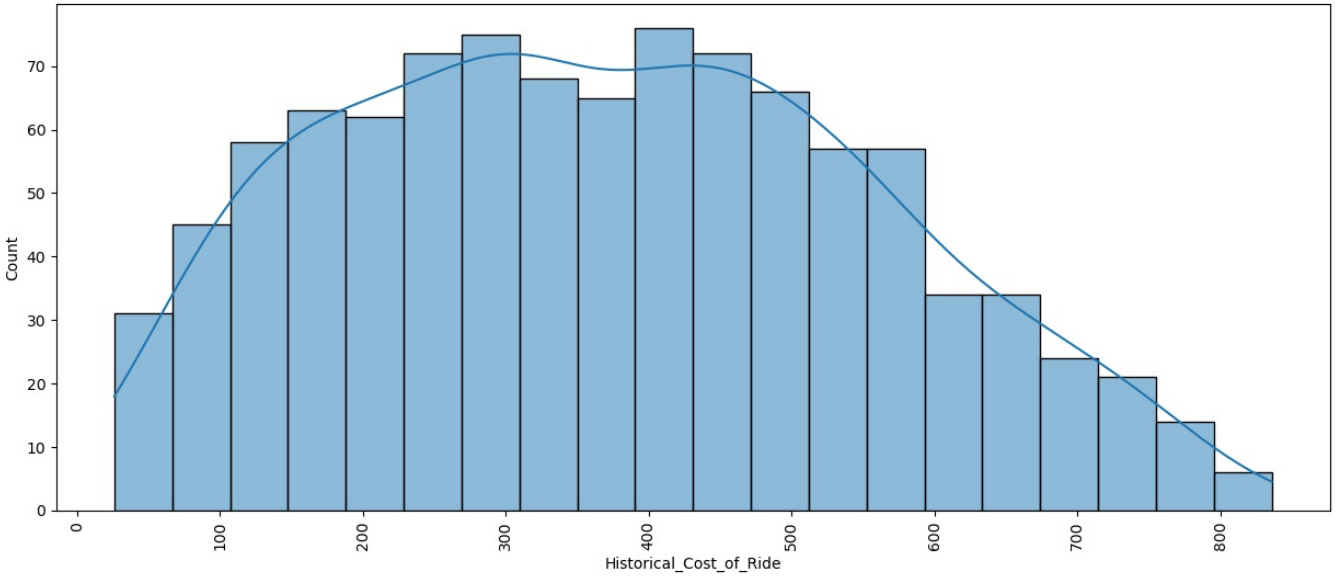

Count
Silver
Regular
Gold
Customer_Loyalty_Status


Count
Number_of_Past_Rides


Count
Average_Ratings







In [33]:
```
from sklearn import preprocessing

# label_encoder object knows
# how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
df['Vehicle_Type']= label_encoder.fit_transform(df['Vehicle_Type'])
```

In [34]:
```
df.head()
```

Out[34]:

	Number_of_Riders	Number_of_Drivers	Location_Category	Customer_Loyalty_Status	Number_of_Past_Rides	Average_Ratings	Time_of_Boo
0	90	45	2	2	13	4.47	
1	58	39	1	2	72	4.06	
2	42	31	0	2	0	3.99	
3	89	28	0	1	67	4.31	
4	78	22	0	1	74	3.77	

In [36]:

```
fig = px.scatter(df, x='Expected_Ride_Duration',
                 y='Historical_Cost_of_Ride',
                 title='Expected Ride Duration vs. Historical Cost of Ride',
                 trendline='ols')
fig.show()
```

In [37]:

```
fig = px.box(df, x='Vehicle_Type',
             y='Historical_Cost_of_Ride',
             title='Historical Cost of Ride Distribution by Vehicle Type')
fig.show()
```

In [38]:
```
corr_matrix = df.corr()

fig = go.Figure(data=go.Heatmap(z=corr_matrix.values,
                                x=corr_matrix.columns,
                                y=corr_matrix.columns,
                                colorscale='Viridis'))
fig.update_layout(title='Correlation Matrix')
fig.show()
```

In [41]:
```
import numpy as np

# Calculate demand_multiplier based on percentile for high and low demand
high_demand_percentile = 75
low_demand_percentile = 25

df['demand_multiplier'] = np.where(df['Number_of_Riders'] > np.percentile(df['Number_of_Riders'], high_demand_p
                                   df['Number_of_Riders'] / np.percentile(df['Number_of_Riders'], high_demand
                                   df['Number_of_Riders'] / np.percentile(df['Number_of_Riders'], low_demand_

# Calculate supply_multiplier based on percentile for high and low supply
```

```
high_supply_percentile = 75
low_supply_percentile = 25

df['supply_multiplier'] = np.where(df['Number_of_Drivers'] > np.percentile(df['Number_of_Drivers'], low_supply_
                                   np.percentile(df['Number_of_Drivers'], high_supply_percentile) / df['Numbe
                                   np.percentile(df['Number_of_Drivers'], low_supply_percentile) / df['Number

# Define price adjustment factors for high and low demand/supply
demand_threshold_high = 1.2  # Higher demand threshold
demand_threshold_low = 0.8  # Lower demand threshold
supply_threshold_high = 0.8  # Higher supply threshold
supply_threshold_low = 1.2  # Lower supply threshold

# Calculate adjusted_ride_cost for dynamic pricing
df['adjusted_ride_cost'] = df['Historical_Cost_of_Ride'] * (
    np.maximum(df['demand_multiplier'], demand_threshold_low) *
    np.maximum(df['supply_multiplier'], supply_threshold_high)
)
```

In [42]:

```
# Calculate the profit percentage for each ride
df['profit_percentage'] = ((df['adjusted_ride_cost'] - df['Historical_Cost_of_Ride']) / df['Historical_Cost_of_
# Identify profitable rides where profit percentage is positive
profitable_rides = df[df['profit_percentage'] > 0]

# Identify loss rides where profit percentage is negative
loss_rides = df[df['profit_percentage'] < 0]


import plotly.graph_objects as go

# Calculate the count of profitable and loss rides
profitable_count = len(profitable_rides)
loss_count = len(loss_rides)

# Create a donut chart to show the distribution of profitable and loss rides
labels = ['Profitable Rides', 'Loss Rides']
values = [profitable_count, loss_count]

fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=0.4)])
fig.update_layout(title='Profitability of Rides (Dynamic Pricing vs. Historical Pricing)')
fig.show()
```

In [43]:

```
fig = px.scatter(df,
                 x='Expected_Ride_Duration',
                 y='adjusted_ride_cost',
                 title='Expected Ride Duration vs. Cost of Ride',
                 trendline='ols')
fig.show()
```

In [44]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

def df_preprocessing_pipeline(df):
    #Identify numeric and categorical features
    numeric_features = df.select_dtypes(include=['float', 'int']).columns
    categorical_features = df.select_dtypes(include=['object']).columns

    #Handle missing values in numeric features
    df[numeric_features] = df[numeric_features].fillna(df[numeric_features].mean())

    #Detect and handle outliers in numeric features using IQR
    for feature in numeric_features:
        Q1 = df[feature].quantile(0.25)
        Q3 = df[feature].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - (1.5 * IQR)
        upper_bound = Q3 + (1.5 * IQR)
        df[feature] = np.where((df[feature] < lower_bound) | (df[feature] > upper_bound),
                               df[feature].mean(), df[feature])

    #Handle missing values in categorical features
    df[categorical_features] = df[categorical_features].fillna(df[categorical_features].mode().iloc[0])

    return df
```

In [45]:
```
from sklearn.model_selection import train_test_split
x = np.array(df[["Number_of_Riders", "Number_of_Drivers", "Vehicle_Type", "Expected_Ride_Duration"]])
y = np.array(df[["adjusted_ride_cost"]])

x_train, x_test, y_train, y_test = train_test_split(x,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=42)

# Reshape y to 1D array
y_train = y_train.ravel()
y_test = y_test.ravel()

# Training a random forest regression model
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
model.fit(x_train, y_train)
```

Out[45]:
```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [46]:
```
x_train
```

Out[46]:
```
array([[ 57,  38,   1, 131],
       [ 21,   8,   1,  84],
       [ 63,  48,   0,  70],
       ...,
       [ 44,  16,   1, 122],
       [ 32,  11,   1,  32],
       [ 58,  34,   1, 131]], dtype=int64)
```

In [47]:
```
y_test
```

Out[47]:
```
array([ 614.28891221,  619.57127194,  694.28149968,  626.10066745,
        338.82519247, 1008.68156984,  255.03496254,  162.86885815,
        698.21957807,  224.76942613,  645.22047413,  305.98205993,
         53.74327221, 1629.41901082,  287.4959047 ,  729.43686971,
        886.86646387,  456.59491663,  302.6887466 ,  980.67662308,
       1032.99596273,   80.814117  ,  251.87576826,  563.68043109,
        604.24071135,  936.92048092,  194.11911277,  978.59941112,
        425.18624191,  882.75210589,  326.70335247,  239.39020469,
        610.15600321,  234.53766778, 1275.10169152,  320.81123118,
        388.10050618,  231.45519848,  402.68097478,  197.73965012,
        988.82161303, 1603.64238812, 1087.73331003, 1267.14216458,
       1152.13123999,  478.55598001,  890.78879568, 1576.02788291,
        381.85215118, 1627.4698265 ,  974.21527229,  455.59167703,
        850.01211175,  551.23814353,  537.48852993,  359.70811179,
        328.84015867, 1588.42985569, 1677.41024587,  630.02236887,
        961.77450852, 1048.53357444,  655.70988464,  814.14342627,
       1362.50330166,  408.36216784,  415.25710035,  289.61375169,
        868.79712859,  200.60572419,  541.971398  ,  287.86995634,
        534.23703899, 1188.48408976,  364.74907464,  511.40982488,
        327.51910669, 2589.31087885, 1467.12691982, 1074.03781862,
        615.73065669,  818.94718556, 2783.68113757,  173.34355137,
        593.23009137,  558.14519434,  520.46058072,   52.75247009,
        939.25874304,  989.89884576,  682.1763357 ,  159.76624109,
        583.80919809, 1077.51903821,  464.68041315, 1128.21670188,
       1297.73488745,  523.80220664,  393.54968001, 1190.12959136,
        356.14389194,  420.16890217,  361.16983283,  506.92705828,
        839.64454639,  483.08999349,  741.57768784,  494.70521414,
        755.04159282,  158.17431075,  551.39347339,  325.41601298,
       1558.41783768,  232.52613103,   40.99340425,   73.81511964,
        319.74946548,  370.11488316, 1487.25430531,  168.01475871,
        803.07387719,  357.35806633,  277.49201735,  242.03504155,
       1050.92133451,  186.73590074,  604.73949707,  584.38204096,
        753.46398454,  182.13274381,  691.69161239,  780.91450395,
       1681.83630925,  124.56789736,  891.83562048,  560.44048888,
        565.19383871,  308.47313565, 1663.75837665,  853.08241647,
        799.29089408,   32.35431252,  930.02114112, 1893.42592846,
        797.98884634, 1073.61821962,  873.35246825,  484.44847009,
        448.942747  ,  394.40755468,  715.67498764, 2125.87409552,
        309.67262421,  248.12369302,  447.56488006,  115.22721107,
        865.84034061, 1511.6299533 ,  341.02410298,   37.77099219,
        941.88821243, 1449.02502043, 1142.34350587,  642.32949335,
        135.15247471, 1475.12225826,  757.14352703, 1051.11778333,
       1558.52362681,  873.80113034,  450.77447417,  567.53183008,
        579.24023191,  427.95290843,  204.20807439,  420.04141416,
        622.03780473,  324.8737472 ,   84.02224207,  333.22868672,
       1254.98022562,  191.12530561,  345.90279857, 1629.16789597,
        996.28443787,  601.4079343 ,  228.93555317,  188.10979948,
       1157.54571441,  120.82066221,  153.41580003,  869.83001313,
       1205.50815107,  789.22540525,  906.40145265, 1015.47119492,
        407.16603141,  862.11804345,  729.76772626,  230.40221427])
```

In [50]:
```
def get_vehicle_type_numeric(vehicle_type):
    vehicle_type_mapping = {
        "Premium": 1,
        "Economy": 0
    }
    vehicle_type_numeric = vehicle_type_mapping.get(vehicle_type)
    return vehicle_type_numeric

# Predicting using user input values
def predict_price(number_of_riders, number_of_drivers, vehicle_type, Expected_Ride_Duration):
    vehicle_type_numeric = get_vehicle_type_numeric(vehicle_type)
    if vehicle_type_numeric is None:
        raise ValueError("Invalid vehicle type")

    input_data = np.array([[number_of_riders, number_of_drivers, vehicle_type_numeric, Expected_Ride_Duration]]
    predicted_price = model.predict(input_data)
    return predicted_price

# Example prediction using user input values
user_number_of_riders = 50
user_number_of_drivers = 25
user_vehicle_type = "Premium"
Expected_Ride_Duration = 30
predicted_price = predict_price(user_number_of_riders, user_number_of_drivers, user_vehicle_type, Expected_Ride
print("Predicted price:", predicted_price)
```

```
Predicted price: [316.15235318]
```

In [49]:
```
import plotly.graph_objects as go
```

```
import plotly.graph_objects as go

# Predict on the test set
y_pred = model.predict(x_test)

# Create a scatter plot with actual vs predicted values
fig = go.Figure()

fig.add_trace(go.Scatter(
    x=y_test.flatten(),
    y=y_pred,
    mode='markers',
    name='Actual vs Predicted'
))

# Add a line representing the ideal case
fig.add_trace(go.Scatter(
    x=[min(y_test.flatten()), max(y_test.flatten())],
    y=[min(y_test.flatten()), max(y_test.flatten())],
    mode='lines',
    name='Ideal',
    line=dict(color='red', dash='dash')
))

fig.update_layout(
    title='Actual vs Predicted Values',
    xaxis_title='Actual Values',
    yaxis_title='Predicted Values',
    showlegend=True,
)

fig.show()
```

In [ ]: